Multilevel Atomicity—A New Correctness Criterion for Database Concurrency Control

NANCY A. LYNCH
Massachusetts Institute of Technology

Multilevel atomicity, a new correctness criteria for database concurrency control, is defined. It weakens the usual notion of serializability by permitting controlled interleaving among transactions. It appears to be especially suitable for applications in which the set of transactions has a natural hierarchical structure based on the hierarchical structure of an organization. A characterization for multilevel atomicity, in terms of the absence of cycles in a dependency relation among transaction steps, is given. Some remarks are made concerning implementation.

Categories and Subject Descriptors: D.3.3 [**Programming Languages**]: Language Constructs—*concurrent programming structures*

General Terms: Design, Performance, Theory

Additional Key Words and Phrases: Transaction, breakpoint, atomicity

1. INTRODUCTION

Popular models for database concurrency control [1, 14] are based on a set of "entities," either centralized or else distributed among the nodes of a network. These entities are accessed by users of the database through "transactions," certain sequences of steps involving the individual entities. The steps are grouped into transactions for at least three distinct purposes. First, a transaction is used as a *logical unit*: it describes a self-contained task within which local state information can persist; thus, the results of earlier steps can be recorded so as to affect the later steps of the same transaction. 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 [1, 3, 14] and has been widely accepted as an important correctness criterion for databases. Third, a transaction is used as a unit of

This work was supported in part by NSF Grant No. MCS79-24370, by U.S. Army Research Office Contract DAAG29-79-C-0155, and by the Advanced Research Projects Agency of the Department of Defense Contract N00014-75-C-0661.

Author's address: Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139.

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.

While the same unit is generally used for all three purposes, I think it is more appropriate to use different units. In particular, the logical unit (henceforth called the "transaction") should be as large as possible, for maximum transaction expressiveness. If transactions are long, then the usual requirement of serializability of transactions is so strong that it excludes efficient implementation of many application databases. Therefore, another mechanism must be superimposed on the transaction mechanism, in order to define atomicity. The unit of atomicity should be as small as possible, for maximum concurrency. The unit of recovery could be anywhere in between; one would probably not want to roll back very long transactions, but might want to roll back beyond a unit of atomicity.

In this paper I consider the simultaneous use of a large logical unit and a smaller unit of atomicity and imagine a database world in which processing is carried out by very long, possibly even infinite, transactions. Each transaction can rely on its memory of previous processing to determine its later processing. From time to time, a transaction reaches a "breakpoint" where other transactions are permitted to interleave. When a transaction resumes processing after a breakpoint, it can recall its activities prior to the breakpoint.

Application databases are modeled here as *centralized, concurrent systems* of transactions and entities. Application databases exist at a purely logical level. Thus, it is appropriate to regard them as centralized even though they are to be "implemented" by a distributed system. 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 *centralized serial 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 would thereby be presented with a view of the account balances which includes the possibility of money being "in transit" from one account to another. I don't think that this interleaving represents an inconvenience to be remedied when technology advances further; rather, it represents the appropriate activity for this application. Between the two extremes, there are many reasonable possibilities.

A framework is required for describing sets of allowable interleavings. Such a framework should specify interleavings in a way which is suitable for use by a concurrency control algorithm. At the same time, the sets of interleavings which can be specified should include 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," 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 is not 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, one might also want to have another type of transaction, an "audit transaction" [4], which would read all of the account balances and return 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 is 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." This definition is probably not general enough to describe all conceivable interesting sets of interleavings. However, it is quite adequate for many applications, and appears especially suited for describing activities of hierarchical organizations. A virtue of this definition is that any set of interleavings thus defined has a simple characterization, in terms of the absence of cycles in a particular dependency relation among transaction steps. This characterization ought to be useful in the design of concurrency control algorithms for multilevel atomicity.

Other researchers [2, 5, 6, 7] have also noted that the usual notion of serializability needs to be weakened. In particular, reference [5] contains interesting preliminary work on specification and concurrency control design, for certain nonserializable interleavings. In fact, the multilevel atomicity of this paper is a generalization of the two-level atomicity described in [5] under the designation "compatibility sets."

The bank transfer-audit example is explored in references [4] and [7]. The solution presented in [4] has the particularly pleasant property that the audit does not stop transactions in progress.

The organization of the rest of the paper is as follows. In Section 2, some examples are given of the sorts of applications for which multilevel atomicity is suited. In Section 3, a formal model is given for application databases. In Section 4, multilevel atomicity is defined. In Section 5, the characterization theorem is stated and proved. Section 6 contains discussion of the possible uses of the characterization theorem for concurrency control design. Section 7 contains discussion of the relationship of multilevel atomicity to the "nested transaction" model of [8, 9, 11] and [13].

Much work remains to be done in designing and evaluating concurrency control algorithms for multilevel atomicity. It remains to be seen 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-than-serializability notions are useful for describing the constraints required for real-world database applications.

2. EXAMPLES

Definitions and claims are illustrated with examples. Many of the illustrations are derived from the following applications.

Application 1. *Banking.* This example expands on the scenarios described in the Introduction. The database for the Big Bucks Bank consists of individual accounts. Bank customers are permitted to manipulate their own accounts in the usual ways. They are also permitted restricted access to the accounts of others (say, to deposit money). As an additional complication for this example, customers are grouped into families, each of which shares control of a common set of accounts. Frequently, a family member will move money between family accounts. Transfers of money from the accounts of one family to the accounts of another family are also fairly common: they are often contingent upon some condition involving the amount of money in one of the originating accounts or the total amount of money in all the accounts of the originating family. Occasionally, the bank wishes to take a complete audit of the contents of all accounts, perhaps using the result to enter a calculated interest amount into a special account. Also, creditors frequently require an audit of the contents of all the accounts of particular families.

The interleaving constraints are very strong for the bank audit: it should be atomic with respect to all the other transactions, and conversely. The interleaving constraints for credit audits and customer transactions are much less severe: for example, as long as the total of the accounts of any particular family is "correct" (e.g., no money is in the process of being moved from one family account to another), it should be possible for any creditor or customer transaction to obtain access to that family's accounts. Finally, the interleaving constraints for customer transactions from customers in the same family are even less severe (perhaps nonexistent). Presumably, family members trust each other enough to allow arbitrary interleaving of accesses to individual accounts (or can be prevailed upon to do so by having to pay less for arbitrarily interleaved service).

It might sometimes be the case that there are some precise database consistency requirements which can be used to determine which interleavings are allowable. For example, the condition that a particular family's total be a correct representation of its assets might be used above to determine where certain interleavings can occur. More usually, however, I expect that such data consistency constraints will be imprecisely understood, very complicated to state, and very difficult to check. I prefer to place emphasis on the transactions themselves rather than on the data. When several transactions are allowed to interleave to a particular degree, I assume it is because they share sufficient understanding of their permitted activities to be willing to allow each other access to some of their partial results. The exact nature of this shared understanding is highly dependent on the semantics of the application.

Application 2. Computer-Aided Design. Utopian Planning, Inc. is an organization which develops detailed plans for design of small cities. The organization consists of a large number of specialized experts: architects, plumbers, traffic engineers, electrical planners, residential-industrial zoning planners, pollution experts, energy efficiency experts, and landscape planners, to name a few. Since there are a large number of experts in some of the categories, these categories are often further subdivided into teams. There is also a public relations department, which has the job of describing the plans to customers intending to build small cities.

Utopian's database for each city consists of the latest plan for that city. All the experts are constantly making changes appropriate to their specialties. These changes interact in very complicated ways. The public relations department requires "snapshots" which describe some reasonable recent version of the plans, satisfying some loosely defined notion of consistency.

Interleaving requirements here are strongest for the snapshots vs. the changes: it is preferred that snapshots be atomic with respect to all changes, and vice versa. Among the changes a large amount of interleaving is allowed; each group of experts expects that the version of the plan on which it begins its work satisfies some minimal consistency constraints required by all the groups of experts. However, this version need not be "sufficiently consistent" to show to customers. Experts within a common specialty share a large body of knowledge about their specialty. Therefore, by agreeing to respect certain consistency constraints appropriate to their specialty, they can permit their changes to interleave to a high degree. Experts within the same team share, in addition to knowledge about their specialty, knowledge about the team's working methods and habits. On this basis, changes made by members of the same team are permitted to interleave to an extremely high degree.

In this example, data-determined consistency constraints would be especially difficult to describe. Nevertheless, it might be easy to describe which groups of transactions "trust each other" to respect appropriate consistency constraints. Note that I have not even described any structure for the database in this example. This structure is extremely complex, and is not required for the approach taken in this paper.

3. APPLICATION DATABASES

In this section we define precise notions of "transaction" and "application database." Application databases consist of a set of transactions together with a set of "correct" interleavings for executions of those transactions. A notion of "equivalence" for transaction executions is defined: two executions are equivalent provided they look the same to each transaction and to each entity in the database. The "correctable" executions are defined to be those which are equivalent to correct executions.

3.1 A Model for Asynchronous Parallel Processes

Application databases will be formalized using a variant of a model [10] for asynchronous parallel computation.

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 a given system, and come equipped with particular initial values. External variables are assumed to be accessible to some "environment" (e.g., other processes or users) which can change the values between steps of a given system.

The computation of a system of processes is described by a set of *executions*. Each execution is a (finite or infinite) totally ordered set of steps which the system could perform when interleaved with appropriate actions by the environment. Each execution consists of steps of the processes of the system.

For any execution e of a system of process, the *dependency partial order*, $\leq_e$, of the steps of e is defined as follows. For every pair of steps, α, β, in e, let $\alpha \leq_e \beta$ if α precedes β in e and either

(1) α and β involve the same process, or
(2) α and β access the same variable.

In this paper I generalize slightly from [10] by allowing executions to be arbitrary totally ordered sets. Therefore, I require the technical assumption that each step in an execution e has only finitely many $\leq_e$ predecessors. The consistency requirements for executions are as follows. Each internal variable starts with its initial value; each execution step involving a process, p, begins with p in the same state which p had at the end of the previous step involving p; each execution step accessing an internal variable, x, begins with x having the same value which x had at the end of the previous step accessing x.

I relax the defintion of "execution" in [10] in one further way, by removing the assumption of fairness. That is, I do not require here that each process continue to take steps until it reaches a final state.

If e is an execution of a system, S, of processes, then every total ordering of the steps of e which is consistent with $\leq_e$ is also an execution of S, having the same sequence of values for each variable and the same sequence of states for each process. We say that two executions, e and e', of S are *equivalent* if $\leq_e$ is identical to $\leq_{e'}$.

3.2 Transactions, Application Databases, Correct and Correctable Executions

My notion of an application database is a centralized, concurrent system consisting of transactions acting on entities, together with a set of correct interleavings of the steps of those transactions. This is modeled very directly in the model in Subsection 3.1: transactions are simply formalized as processes, while entities are formalized as variables. More precisely, an *application database* (S, C) consists of a system S of processes, where all variables of S are internal (i.e., internal to the system), together with a subset C of the executions of S. The processes are called *transactions*, while the variables are called *entities*. The elements of C are called *correct* executions. The assumption that the variables are internal means that the entities are only accessed via the transactions.

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 [5] are a particular type of nondeterministic transaction in the present model.

If (S, C) is an application database and e is an execution of S, we say that e is *correctable* provided e is equivalent to some $e' \in C$.

Example. If C is the set of serial executions of the transaction system [3], then the correctable executions are just the usual serializable executions.

4. MULTILEVEL ATOMICITY

4.1 Motivation

One would like to be able to define particular application databases and have a (centralized or distributed) system able to "implement" them. That is, the system should "simulate" (in some sense which will remain unspecified) only correctable executions for the transactions. For arbitrary choices of C, this task could be very difficult.

For the case where C is the set of serial executions, concurrency control theory provides help. A basic theorem [1, 3] characterizes the serializable executions as those having an absence of cycles in a certain relation describing dependencies among transactions. Thus, one can ensure serializability by explicitly preventing unwanted cycles (using such devices as two-phase locking [3] and timestamps [7]).

In this section I restrict the form of C so that a similar cycle-free characterization can be obtained. The particular method of restriction I use is to group transactions into nested classes. Those which are more closely related in the nesting structure will be permitted to interleave at a finer level of atomicity. This structure has the advantage that it allows breakpoint specifications for each transaction to be given solely in terms of nesting level. Nested classes are appropriate for describing the examples given in Section 2, and for describing other examples which model activities of hierarchical organizations.

4.2 Coherent Relations

The definitions in this subsection are presented at an abstract level (using sets and partial orders) because they are used to prove a general combinatorial lemma in Subsecton 5.1.

A *k-nest*, π, for a set X assigns an equivalence relation $\pi(i)$ to each i, $1 \leq i \leq k$, in such a way that

(1) $\pi(1)$ consists of exactly one equivalence class,
(2) $\pi(k)$ consists of singleton equivalence classes, and
(3) each $\pi(i)$ is a (not necessarily proper) refinement of its predecessor, $\pi(i-1)$.

If $x, x' \in X$, then $\text{level}_\pi(x, x')$ denotes the largest i for which $(x, x') \in \pi(i)$(that is, for which x and x' are related by equivalence relation $\pi(i)$). Thus, pairs with higher-numbered levels are more closely related.

We will consider cases where X is a set of transactions, as in the following two examples.

Example (*Banking*). The set X consists of customer transactions, bank audit transactions, and creditor transactions. A 4-nest describes the relevant relationships among transactions. $\pi(1)$ relates all the transactions. $\pi(2)$ relates all customer and creditor transactions and places each bank audit transaction in a singleton class. $\pi(3)$ refines $\pi(2)$ by relating only those customer transactions belonging to a common family. $\pi(4)$ consists entirely of singleton classes.

Example (*Computer-Aided Design*). The set X consists of snapshot transactions and modification transactions. A 5-nest describes the important relationships. $\pi(1)$ relates all the transactions. $\pi(2)$ groups all modification transactions together and all snapshot transactions together. $\pi(3)$ refines $\pi(2)$ by relating only those modification transactions belonging to a common specialty, and $\pi(4)$ refines $\pi(3)$ by relating only those belonging to a common team. Finally, $\pi(5)$ consists of singleton classes.

Next, I describe sets of breakpoints within a totally ordered set, one set of breakpoints for each of several "levels," in such a way that the higher level sets of breakpoints always include the lower level sets. The totally ordered set should be thought of as the set of steps of some execution of a particular transaction.

If $(X, \leq)$ is a total order, then an equivalence relation, $\equiv$, on X is said to be a $\leq$-segmentation, provided that $\alpha \equiv \beta$ and $\alpha \leq \gamma \leq \beta$ together imply $\alpha \equiv \gamma$. That is, each equivalence class is a segment consisting of consecutive elements of X.

Breakpoints will be described formally by describing the segments between the breakpoints, as follows. Once again, a k-nest (this time for the steps of the transaction) is useful. If $(X, \leq)$ is a total order, then a *k-level breakpoint description*, B, for $(X, \leq)$ is a k-nest for X such that each $B(i)$ is a $\leq$-segmentation.

Example (*Banking*). Let the elements of $(X, \leq)$ be $\omega_1, \omega_2, \omega_3, \delta_1, \delta_2$, in $\leq$ order. Then B given as follows is a 4-level breakpoint description for $(X, \leq)$:

$B(1)$'s only class is $\{\omega_1, \omega_2, \omega_3, \delta_1, \delta_2\}$,
$B(2)$'s classes are $\{\omega_1, \omega_2, \omega_3\}$ and $\{\delta_1, \delta_2\}$,
$B(3)$'s classes are $\{\omega_1\}$, $\{\omega_2\}$, $\{\omega_3\}$, $\{\delta_1\}$ and $\{\delta_2\}$, and
$B(4)$ is identical to $B(3)$.

Intuitively, $\omega_1, \omega_2, \omega_3, \delta_1, \delta_2$ might represent the sequence of steps of a particular execution of a funds-transfer transaction. Steps ω_1, ω_2, and ω_3 represent withdrawals from accounts belonging to the family originating the transaction. The amounts obtained by these withdrawals depend on the amounts which are discovered to be in the accounts. Steps δ_1 and δ_2 represent deposits to two arbitrary other accounts (say, a fuel bill account and an entertainment account). The amounts deposited in the two accounts might depend on the amount discovered to be already in the first account. $B(1)$ and $B(4)$ just represent the extreme cases of atomicity. $B(2)$ represents the breakpoint (between ω_3 and δ_1) where other customer and creditor transactions (but not bank audit transactions) are permitted to interleave. $B(3)$ represents the breakpoints permitted for other transactions of the same family as the given funds-transfer transaction.

Next, I want to describe sets of breakpoints for all the transactions in a given set. If T is a set (to be thought of as a set of transactions), then a *k-level interleaving specification*, $\mathscr{I}$, for T, is a collection of triples $(X_t, \leq_t, B_t)$, one for each $t \in T$, where $\{(X_t, \leq_t) : t \in T\}$ is a collection of disjoint totally ordered sets (to be thought of as the sets of steps of particular executions of all the transactions in T) and each B_t is a k-level breakpoint description for $(X_t, \leq_t)$.

Example (*Banking*). Let $T = \{t_1, t_2, t_3\}$. For each t_i, let $(X_{t_i}, \leq_{t_i})$ be the sequence $\omega_{i1}, \omega_{i2}, \omega_{i3}, \delta_{i1}, \delta_{i2}$, and let B_{t_i} be defined by analogy to the previous example:

$B_{t_i}(1)$'s only class is $\{\omega_{i1}, \omega_{i2}, \omega_{i3}, \delta_{i1}, \delta_{i2}\}$,
$B_{t_i}(2)$'s classes are $\{\omega_{i1}, \omega_{i2}, \omega_{i3}\}$ and $\{\delta_{i1}, \delta_{i2}\}$, etc.

Then $\mathscr{I} = \{(X_t, \leq_t, B_t) : t \in T\}$ is a 4-level interleaving specification for T.

Intuitively, t_1, t_2, and t_3 represent different funds-transfer transactions, which might be from the same or different families. $\mathscr{I}$ gives both a sequence of steps and a breakpoint description for each of t_1, t_2, t_3. This combination of descriptions is intended to be used to help define how t_1, t_2, and t_3 are permitted to interleave. (Of course, in order to define the permissible interleavings, we must also know which of t_1, t_2, and t_3 are from common families.)

Next, I define an important condition for a relation, R, on $\cup \{X_t : t \in T\}$. I want to express the fact that R preserves all of the individual $\leq_t$ orderings and also respects the restrictions expressed by the given collection of breakpoint descriptions. In most cases of interest, R will be a partial order.

Let π be a k-nest for T, $\mathscr{I} = \{(X_t, \leq_t, B_t) : t \in T\}$ a k-level interleaving specification for T, and R a relation on $\cup \{X_t : t \in T\}$. Then R is *coherent* for π and $\mathscr{I}$ provided the following two conditions hold.

(1) R contains each partial order $\leq_t$.
(2) Assume $\text{level}_\pi(t, t') = i$.
    Assume $\alpha, \alpha' \in X_t$ and $\alpha \leq_t \alpha'$ and $(\alpha, \alpha') \in B_t(i)$.
    Assume $\beta \in X_{t'}$.
    If $(\alpha, \beta) \in R$, then $(\alpha', \beta) \in R$.

Intuitively, this latter condition says the following. If a step, β, of one transaction follows (in R) a step, α, of another transaction, t, then β also follows any other step, α', of t which follows α but precedes any breakpoints of the appropriate level. Note that the breakpoints are defined solely in terms of the nesting level for the two transactions.

Example. Let $k = 3$, $T = \{t_1, t_2, t_3\}$ and let $\pi(2)$'s classes be $\{t_1, t_2\}$ and $\{t_3\}$. ($\pi(1)$ and $\pi(3)$ are uniquely determined.) For each $t_i \in T$, let $(X_{t_i}, \leq_{t_i})$ be the sequence $\alpha_{i1}, \alpha_{i2}, \alpha_{i3}, \alpha_{i4}$, and let $B_{t_1}(2)$'s classes be $\{\alpha_{i1}, \alpha_{i2}\}$ and $\{\alpha_{i3}, \alpha_{i4}\}$. ($B_{t_i}(1)$ and $B_{t_i}(3)$ are uniquely determined.)

Let R_1 be the transitive closure of all the $\leq_{t_i}$ plus the pairs $(\alpha_{12}, \alpha_{22})$, $(\alpha_{22}, \alpha_{13})$, $(\alpha_{14}, \alpha_{31})$ and $(\alpha_{24}, \alpha_{33})$. Then R_1 is a coherent partial order.

Let R_2 be the transitive closure of all the $\leq_{t_i}$ plus the pairs $(\alpha_{11}, \alpha_{22})$, $(\alpha_{21}, \alpha_{13})$, $(\alpha_{11}, \alpha_{31})$ and $(\alpha_{21}, \alpha_{33})$. Then R_2 is a noncoherent partial order.

Let R_3 be constructed similarly to R_2, except with $(\alpha_{31}, \alpha_{11})$ in place of $(\alpha_{11}, \alpha_{31})$. Then R_3 is a noncoherent partial order.

If a given relation R is not coherent, it is sometimes useful to consider the smallest coherent relation containing R. This can be defined as follows. Given a set T, a k-nest π for T, a k-level interleaving specification $\mathscr{I} = \{(X_t, \leq_t, B_t): t \in T\}$ for T, and a relation R on $\cup \{X_t: t \in T\}$ containing all the $\leq_t$, define the *coherent closure* of R with respect to π and $\mathscr{I}$ to be the relation obtained from R by closing under Condition (b) of the coherence definition.

Example. In the previous example, the coherent closure of R_1 is R_1 itself. The coherent closure of R_2 is just the partial order R_1. The coherent closure, R_4, of R_3 is not a partial order, however. (Since $(\alpha_{31}, \alpha_{11}) \in R_4$, it follows that $(\alpha_{33}, \alpha_{11}) \in R_4$. We know $(\alpha_{11}, \alpha_{22}) \in R_4$. Since $(\alpha_{21}, \alpha_{33}) \in R_4$, it follows that $(\alpha_{22}, \alpha_{33}) \in R_4$. Hence, R_4 contains a cycle.)

It is easy to see that R is extendable to a coherent partial order if and only if the coherent closure of R is a partial order.

4.3 Definition of Multilevel Atomicity

In contrast to the preceding subsection, the definitions in this subsection deal explicitly with a system S of transactions. I use the abstract definitions in Section 3 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, e, of t can be interrupted by steps of t'.

A *k-level breakpoint specification*, $\mathscr{B}$, for a system, S, of transactions is a family, $\{B_{t,e}: t$ is a transaction of S, e an execution of $t\}$, where each $B_{t,e}$ is a k-level breakpoint description 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, α_i), where α_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. Namely, for any execution e of S, define a k-level interleaving specification, $\mathcal{I}(\mathcal{B}, e) = \{(X_t, \leq_t, B_t): t \in T\}$, by letting T be the set of transactions appearing in e, e_t be the execution of t occurring as a subsequence of e, X_t be the set of steps of t occurring in e_t, $\leq_t$ be the order in which those steps occur in e, and B_t be $B_{t,e_t} \in \mathcal{B}$. $\mathcal{I}(\mathcal{B}, e)$ is just the natural interleaving specification which is derived from the particular execution e using the given k-level breakpoint specification $\mathcal{B}$. An execution e of S is *multilevel atomic* for π and $\mathcal{B}$ provided the total ordering of steps in e is coherent for π and $\mathcal{I}(\mathcal{B}, e)$. Let $C(\pi, \mathcal{B})$ denote the set of executions which are multilevel atomic for π and $\mathcal{B}$. Then the application database of interest is $(S, C(\pi, \mathcal{B}))$.

Thus, we use the multilevel atomic executions as the "correct" executions. In Section 5, we develop a characterization of the corresponding "correctable" executions. Note that "multilevel atomic" generalizes "serial," as follows.

Example. If $k = 2$, then $\pi(1)$ relates all transactions, while $\pi(2)$ only relates transactions to themselves. There is only one possible breakpoint specification $\mathcal{B}$. Namely, for each t and e, $B_{t,e}(1)$ groups all steps together, while $B_{t,e}(2)$ divides the steps into singleton sets. In this case, the multilevel atomic executions are just the serial executions.

Example. The reader is referred to [5] for treatment of a special case of our definition corresponding to $k = 3$, where $B_{t,e}(2)$ consists of single steps, for all t and e. That is, transactions in a common $\pi(2)$ class can interleave arbitrarily, but transactions not in a common $\pi(2)$ 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 [5].

Example (Banking). Let the set of transactions be $T \cup A$, where $T = \{t_1, t_2, t_3\}$ is a set of transfers and $A = \{a\}$ consists of a single bank audit. Let π be the 4-nest with $\pi(2) = \{t_1, t_2, t_3\}, \{a\}$ and $\pi(3) = \{t_1, t_2\}, \{t_3\}, \{a\}$.

Consider t_1, for example. It is intended to withdraw \$100 from the combined accounts A, B, and C, and deposit the withdrawn amount in D and E. The precise behavior of t_1 depends on the amounts encountered in the various accounts. It will examine A, B, and C sequentially, attempting to obtain \$100 as soon as possible. If t_1 is able to obtain \$100 from A alone or from just A and B, then it need not access the remaining accounts. If t_1 accesses all three accounts and succeeds in obtaining less than \$100, it will proceed to D and E with the lesser amount. It tries to leave D with at least \$125; any available money over \$125 will be deposited in E.

Thus, t_1 has many possible execution sequences. Two are described below.

e_1: Access A, see \$20, leave \$0.
Access B, see \$150, leave \$70.
Access D, see \$20, leave \$120.

e_2: Access A, see \$0, leave \$0.
Access B, see \$15, leave \$0.
Access C, see \$70, leave \$0.
Access D, see \$110, leave \$125.
Access E, see \$30, leave \$100.

Let $\mathscr{B} = \{B_{t,e}: t \in T \cup A,\ e \text{ an execution for } t\}$ be the 4-level breakpoint specification for $T \cup A$ defined as described in the banking examples in Subsection 4.2. For example, $B_{t_1,e_2}(2)$ has classes $\{\omega_1, \omega_2, \omega_3\}$, $\{\delta_1, \delta_2\}$, where ω_1, ω_2, ω_3, δ_1, and δ_2 represent the five steps of e_2, in sequence. (For all transfers, $B_{t,e}(2)$ groups withdrawal steps together and deposit steps together.) $B_{t_1,e_2}(3)$ consists of singleton classes.

Now, for each t_i, fix a corresponding execution e_i with steps ω_{i1}, ω_{i2}, δ_{i1}, δ_{i2}. Fix an execution e of a with steps α_1, α_2, α_3. If the following is an execution (i.e., if the successive values of entities match up properly), then it is multilevel atomic for π and $\mathscr{B}$:

$$\omega_{31}, \omega_{32}, \omega_{11}, \omega_{21}, \omega_{22}, \omega_{12}, \delta_{31}, \delta_{32}, \delta_{21}, \delta_{11}, \delta_{22}, \delta_{12}, \alpha_1, \alpha_2, \alpha_3.$$

5. CHARACTERIZATION THEOREM

In the previous section, a particular style of definition for C, the set of correct sequences, was given. One would like a centralized or distributed processing system to "simulate" only correctable executions. ("Simulation," as already mentioned, is not used in the context of any precise definition.) In this section a characterization theorem is proved for correctable executions. This theorem is analogous to the absence-of-cycles characterization for serializability [3].

5.1 A Combinatorial Lemma

In this subsection I state a combinatorial lemma which will be used in the next section to derive a necessary and sufficient condition for correctability (equivalence with multilevel atomicity). The lemma requires only the abstract definitions in Subsection 4.2.

For this subsection, let T be a fixed set, let π be a fixed k-nest for T, and let $\mathscr{I} = \{(X_t, \leq_t, B_t): t \in T\}$ be a fixed k-level interleaving specification for T. Let "coherent" mean "coherent for π and $\mathscr{I}$," and write "level" for "level$_\pi$".

LEMMA 1. *If $\leq$ is a coherent partial order, then there is a coherent total order $\leq'$ which contains $\leq$.*

PROOF. See Appendix.

Example. Let R_1 be the coherent partial order given in Subsection 4.2. Then there are two coherent total orders containing R_1:

$$\alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \alpha_{13}, \alpha_{14}, \alpha_{23}, \alpha_{24}, \alpha_{31}, \alpha_{32}, \alpha_{33}, \alpha_{34},$$

and

$$\alpha_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \alpha_{23}, \alpha_{24}, \alpha_{13}, \alpha_{14}, \alpha_{31}, \alpha_{32}, \alpha_{33}, \alpha_{34}.$$

5.2. The Theorem

The characterization result can now be stated. For this subsection, let S be a fixed set of transactions, π a fixed k-nest for S, and $\mathscr{B}$ a fixed k-level breakpoint specification for S. Let the "correct" executions denote those in $C(\pi, \mathscr{B})$(i.e., the multilevel atomic executions), and the "correctable" executions denote those which are equivalent to multilevel atomic executions.

THEOREM 1. *Let e be an execution of S. Then e is correctable if and only if the coherent closure of $\leq_e$ with respect to π and $\mathcal{I}(\mathcal{B}, e)$ is a partial order.*

PROOF. First, assume e is correctable. This means that $\leq_e$ is extendable to a total order which is in $C(\pi, \mathcal{B})$, that is, which is coherent for π and $\mathcal{I}(\mathcal{B}, e)$. Then surely the coherent closure of $\leq_e$, which is the smallest coherent relation containing $\leq_e$, must be acyclic.

Conversely, assume that the coherent closure of $\leq_e$ with respect to π and $\mathcal{I}(\mathcal{B}, e)$ is a partial order. Then the lemma implies that there is a coherent (for π and $\mathcal{I}(\mathcal{B}, e)$) total order which includes the coherent closure of $\leq_e$, and which therefore includes $\leq_e$. Thus, e is correctable. □

Example (*Banking*). Consider the last example in Subsection 4.3, where the transactions are t_1, t_2, t_3 and a, and fix executions as before. Assume the accounts accessed are as follows.

ω_{11}:A	ω_{21}:A	ω_{31}:B	α_1:A
ω_{12}:B	ω_{22}:C	ω_{32}:D	α_2:B
δ_{11}:C	δ_{21}:E	δ_{31}:F	α_3:C
δ_{12}:D	δ_{22}:G	δ_{32}:H	

If the following is an execution, then, while it is not multilevel atomic for π and $\mathcal{B}$, it is correctable:

$$\omega_{11}, \omega_{31}, \omega_{21}, \omega_{12}, \alpha_1, \alpha_2, \omega_{22}, \delta_{11}, \alpha_3, \delta_{21}, \delta_{22}, \omega_{32}, \delta_{12}, \delta_{31}, \delta_{32}.$$

An equivalent multilevel atomic execution is the one given in Subsection 4.3.

On the other hand, if the following is an execution, then it is not correctable:

$$\omega_{11}, \omega_{21}, \omega_{31}, \alpha_1, \alpha_2, \alpha_3, \omega_{12}, \omega_{22}, \omega_{32}, \delta_{11}, \delta_{21}, \delta_{31}, \delta_{12}, \delta_{22}, \delta_{32}.$$

The theorem can be used to verify both claims. For instance, to see that the second execution is not correctable, we describe a particular cycle in the coherent closure, namely, $\alpha_3 < \omega_{22} < \alpha_1 < \alpha_3$, where the second inequality follows when we consider that $\omega_{21} < \alpha_1$ and take the coherent closure.

6. CONCURRENCY CONTROL

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 condition stated in the theorem in Section 5.

For definiteness, I use the "migrating transaction" model described in [14]. 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 p. A message (p, t, s) is sent to the processor owning the entity which t accesses when it is in state s. A processor receiving a message (p, t, s) "performs" the indicated step by changing the value of the entity, updating t's state, and sending a new message (p, t, s'), where s' is the new state. If s' is not a final state, the messages is sent to the processor owning the appropriate entity. If s' is a final state, the message is sent back to the originator p. In this way, an execution e of the system of transactions is actually "performed" by the processors. The total order of the execution is determined by real clock time.

I consider how to insure that any execution sequence e "performed" by the processors has a dependency partial order $\leq_e$ whose coherent closure is a partial order.

It will be necessary to make an additional assumption about a breakpoint specification—in order to be able to determine the locations of breakpoints on-line, it is necessary to assume a "compatibility" condition: if two executions of a transaction share a common prefix $\bar{e}$, then either both executions have a breakpoint immediately after $\bar{e}$, or neither does.

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

One possible strategy is cycle detection, using the coherent closure of $\leq_e$. If the concurrency control does not otherwise guarantee that $\leq_e$ is extendable to a coherent partial order, the concurrency control might explicitly generate the edges of the coherent closure of $\leq_e$, 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 strict serializability were required, leading to fewer rollbacks.

Another approach is cycle prevention: guaranteeing that the coherent closure of $\leq_e$ is a partial order. One way of doing this might be to delay some steps, as follows.

Let β be a step of any transaction t'. β first gets "scheduled," thereby locking its entity and delaying t'. (The lock on the entity prevents any other step from being performed on it.) β does not actually get "performed" until the following is ensured. (Note that e refers to the order in which steps actually get performed, not the order in which they are scheduled.) Let e_β denote the initial segment of e ending with step β. If α is the last step of some transaction t which precedes β in the coherent closure of $\leq_{e_\beta}$, then a level(t, t') breakpoint immediately follows α in the execution subsequence of e_β belonging to t. (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 $\leq_e$ 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 is that even though the concurrency control guarantees eventual performance of all of the steps of a correct execution 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, there are steps α of t_i and β of t_{i+1} with $\beta \leq_e \alpha$. 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.

7. DISCUSSION

7.1. Nested Transactions

It is interesting to compare multilevel atomicity to the atomicity achieved by the "nested transaction model" [9, 11, 13]. The latter model permits transactions to be nested, and then requires serializability of transactions at every level, including the top level. The remainder of this subsection assumes familiarity with some of the work on nested transactions.

At first glance, it appears that the nested transaction model is incapable of describing the interleavings considered in this paper. Indeed, this is the case if the atomicity units ("transactions" of the nested transaction model) are constrained to be the same as the logical units ("transactions" of this paper).

Example (*Banking*). Let T be a set of transfer transactions, A a set of audit transactions. If each element of $T \cup A$ is modeled as a separate top-level transaction in the nested transaction model, then elements of T are required to be serialized with respect to each other.

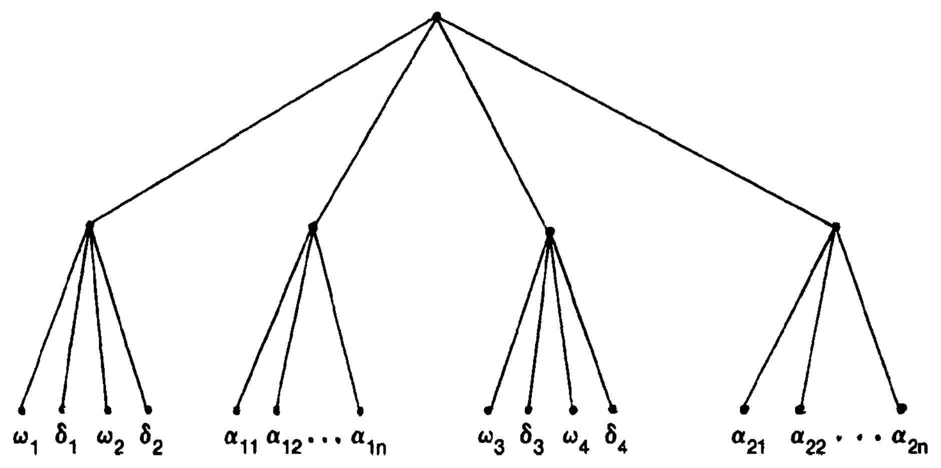
However, the situation is different if the logical units and the units of atomicity are allowed to be different. The nested "transactions" of the nested transaction model can be regarded as describing the units of atomicity.

In order to distinguish these from the logical transactions, I will designate the former as "actions." A (logical) transaction would be mapped into actions by means of a mapping which distorts the transaction's structure.

Example (*Banking*). Let $T = \{t_1, \ldots, t_4\}$ be a set of transfers, where each transfer t_i consists of a withdrawal step ω_i followed by a deposit step δ_i. Let $A = \{a_1, a_2\}$ be a set of audits, where each audit a_i consists of a sequence $\alpha_{i1}, \ldots, \alpha_{in}$ of read-account-balance steps. A nested action tree can be used to describe the relevant nesting relationships between actions for each multilevel atomic execution. For example, the tree in Figure 1 can be used to describe an execution in which transactions t_1 and t_2 are combined to form a single action. The steps of the two transactions, ω_1, δ_1, ω_2 and δ_2 are all siblings as far as atomicity is concerned.

There are several possible ways in which ω_1, δ_2, ω_2 and δ_2 might interleave. Similarly, t_3 and t_4 are combined. (Note that the reorganization of transactions into actions is not statically determined, but rather depends on the particular execution.) With this reorganization, the nested transaction model expresses exactly the proper atomicity requirements.

In a way similar to that described in the preceding example, any set of multilevel atomic executions $C(\pi, \mathscr{B})$ can be described by a corresponding collection of nested action trees. In each such nested action tree, the following property holds.



ω_1 δ_1 ω_2 δ_2
α_{11} $\alpha_{12}\cdots\alpha_{1n}$
ω_3 δ_3 ω_4 δ_4
α_{21} $\alpha_{22}\cdots\alpha_{2n}$

Fig. 1

Enumerate the levels of the tree, with the root at level 1. Then all steps appearing below any particular level i node in the tree belong to transactions which are $\pi(i)$-equivalent. Moreover, (if $i > 1$), these steps suffice to carry each of the transactions involved to a level $i - 1$ breakpoint. In this way, the nested action tree structure follows the k-nest structure.

Although it is possible for the nested transaction model to describe multilevel atomicity, it is not clear to what extent this fact is useful for implementing multilevel atomicity. There are several known ways of implementing nested transactions, based on timestamps [13] or two-phase locking [8, 11]. Of course, these could be specialized to implement multilevel atomicity. However, I do not know whether these specializations provide efficient implementations. This question is a topic for future study.

The new programming language Argus [8] is based on the nested transaction model. In that language, the structure of user programs follows the nested action structure very closely. That is, the logical unit and the unit of atomicity are the same. While I suspect that the nested transaction model is adequate for describing atomicity, it seems to me that for modeling many situations of interest (multilevel atomicity, conversations between transactions [12]), it will be necessary for the logical program structure to be different from the atomicity structure. Perhaps both logical structure and atomicity are naturally described using nested structures, but the nestings used for those two purposes might be different.

7.2 Remaining Questions

There are several areas remaining for future research: exploration of more applications in which multilevel atomicity is a helpful descriptive tool; the design of detailed concurrency controls based on this criterion, and the use of them to determine whether this generalization of serializability can be exploited for increased efficiency are a few such areas. It also remains to be seen whether implementation of multilevel atomicity as a special case of the nested transaction model will provide reasonable efficiency.

Of the greatest and most general significance, though, is the identification of other situations in which it is useful to distinguish the logical unit from the unit of atomicity (and from the unit of recovery).

APPENDIX. PROOF OF THE COMBINATORIAL LEMMA

Let $\leq^{(1)}$ denote $\leq$. A sequence of *stages* numbered $2, \ldots, k$ 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 $\alpha \in X_t$ and $\beta \in X'_t$, and level$(t, t') <$, 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 $X = \cup \{X_t : t \in T\}$ into *segments*, where each segment is an equivalence class of some $B_t(i-1)$.

A segment S is said to *belong to* an element $t \in T$ if $S \subseteq X_t$.

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, $\mathcal{S}_1 \leq \mathcal{S}_2 \leq \ldots$, 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 $m < n$. (Stage i is now complete.)

I now prove the needed properties (a) and (b) for $\leq^{(i)}$, assuming that they hold for $\leq^{(i-1)}$. (Note, there is no Lemma 2.)

LEMMA 3. *$\leq^{(i)}$ is a partial order.*

PROOF. 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 $\leq^{(i)}$. □

LEMMA 4. *$\leq^{(i)}$ is coherent.*

PROOF. Assume level$(t, t') = j$. Assume $\alpha, \alpha' \in X_t$ and $\alpha \leq_t \alpha'$ and $(\alpha, \alpha') \in B_t(j)$. Assume $\beta \in X'_t$. Assume $\alpha \leq^{(i)} \beta$. I show that $\alpha' \leq^{(i)} \beta$. The result is trivial if $t = t'$, so assume that $t \neq t'$.

Case 1. $\alpha \leq^{(i-1)} \beta$. By inductive hypothesis (a), $\leq^{(i-1)}$ is coherent, which implies the needed result.

Case 2. $\alpha \not\leq^{(i-1)} \beta$. 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 $\beta \not\leq^{(i-1)} \alpha$, so that α and β are $\leq^{(i-1)}$-incomparable. Then inductive hypothesis (b) implies that j (= level(t, t')) $\geq i - 1$. Thus, $B_t(j) \subseteq B_t(i-1)$(i.e., $B_t(j)$ is a refinement of $B_t(i-1)$), so that $(\alpha, \alpha') \in B_t(i-1)$. Therefore, $\alpha' \in S_1$. The definition of $\leq^{(i)}$ then insures the needed result. □

LEMMA 5. *For each m, the following holds. If $S, S' \in \mathcal{S}_m$, S belongs to t and S' belongs to t', then $(t, t') \in \pi(i)$.*

PROOF. If not, then some $\mathcal{SS}_m$ contains a cycle $S_0, S_1, \ldots, S_l = S_0$ of segments such that for each j, $0 \le j \le l-1$, there exist $\alpha \in S_j$, $\beta \in S_{j+1}$ with $\alpha \le^{(i-1)} \beta$ and such that two of the segments belong to $\pi(i)$-inequivalent elements of T.

Let S and S' be two distinct segments in this cycle, belonging to elements t and t' respectively, where (1)$(t, t') \notin \pi(i)$, and (2) any segment S'' following S and preceding S' in the cycle belongs to some t'' which is $\pi(i)$-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. (This is by construction of the cycle and the fact that $\le^{(i-1)}$ contains all the total orderings of the individual transactions.) By inductive hypothesis (or trivially, if S' itself is the 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 $B_t(i-1) \subseteq B_t(j)$. Since $(\alpha', \alpha) \in B_t(i-1)$, it follows that $(\alpha', \alpha) \in B_t(j)$. Coherence of $\le^{(i-1)}$ implies that $\alpha \le^{(i-1)} \beta$.

Applying this claim 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. □

LEMMA 6. *If $\alpha \in X_t$ and $\beta \in X_{t'}$, and level$(t, t') < i$, then α and β $\le^{(i)}$-comparable.*

PROOF. By Lemma 5, 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 $m \ne n$. But then $\le^{(i)}$ is defined to contain the pair (α, β) if $m < n$, and to contain (β, α) if $n < m$. □

ACKNOWLEDGMENTS

The author is grateful to Nancy Griffeth, Mike Merritt, and Mike Fischer for many discussions about the subjects covered in this paper. Comments and suggestions from Phil Bernstein, Nat Goodman, and Bill Weihl are also appreciated.

REFERENCES

1. BERNSTEIN, P. A., AND GOODMAN, N. Concurrency control in distributed database systems. *ACM Comput. Surv. 13*, 2 (June 1981), 185–221.
2. CLARK, D. Oral presentation at the Fifth Berkeley Workshop on Distributed Data Management and Computer Networks, Feb. 3–5, 1981.
3. ESWAREN, K. P., GRAY, N. N., LORIE, R. A., AND TRAIGER, I. L. The notions of consistency and predicate locks in a database system. *Commun. ACM 19*, 11 (Nov. 1976), 624–633.
4. FISCHER, M. J., GRIFFETH, N. D., AND LYNCH, N. A. Global states of a distributed system. In *Proc. IEEE Symp. Reliability in Distributed Software and Database Systems*, July 1981. Also in *IEEE Trans. Softw. Eng. SE 8*, 3 (May 1982), 198–202.
5. GARCIA-MOLINA, H. Using semantic knowledge for transaction processing in a distributed database. Tech. Rep. 285, Princeton Univ. Dept. of Electrical Engineering and Computer Science, April 1981.
6. GRAY, J. N., LORIE, R. A., PUTZOLU, G. R., AND TRAIGER, L. I. Granularity of locks and degrees of consistency in a shared database. In *Proc. IFIP Working Conf. on Modelling of Data Base*

Management Systems (Freudenstadt, Germany, Jan. 1976), pp. 695–723. Also in *Modelling in Data Base Management Systems,* G. M. Nijssen, Ed., Elsevier North-Holland, 1976, pp. 365–395.

7. LAMPORT, L. Towards a theory of correctness of multi-user database systems. Massachusetts Computer Associates, CA-7610-0712, Oct. 1976.
8. LISKOV, B., AND SCHEIFLER, R. Guardians and actions: linguistic support for robust, distributed programs. In *1982 Ninth Annual ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages* (Albuquerque, Jan. 25–27, 1982) pp. 7–19.
9. LYNCH, N. A. Concurrency control for resilient nested transactions. MIT/LCS/TR-285, MIT, Laboratory for Computer Science, Feb. 1983.
10. LYNCH, N. A., FISCHER, M. J. On describing the behavior and implementation of distributed systems. *Theor. Comput. Sci. 13,* (Jan. 1981), 17–43.
11. MOSS, J. E. B. Nested transactions: an approach to reliable distributed computing. PhD Dissertation, Tech. Rep. MIT/LCS/TR-260, MIT, Laboratory for Computer Science, Cambridge, Mass., 1981.
12. RANDELL, B. System structures for software fault tolerance. In *Proc. Int. Conf. on Reliable Software* (Los Angeles, April 21–23, 1975), ACM, New York, 1975, pp. 437–457. Also in *SIGPLAN Notices 10,* 6 (June 1975); also in *IEEE Trans. Softw. Eng. 1,* 2 (June 1975), 220–232.
13. REED, D. P. Naming and synchronization in a decentralized computer system. PhD Dissertation, Tech. Rep. MIT/LCS/TR-205, MIT, Laboratory for Computer Science, Cambridge, Mass., 1978.
14. ROSENKRANTZ, D., STEARNS, R., AND LEWIS, P. System level concurrency control for distributed database systems. *ACM Trans. Database Syst. 3,* 2 (June 1978), 178–198.

Received June 1981; revised September 1982; accepted November 1982