Relative Serializability: An Approach for Relaxing the Atomicity of Transactions

D. Agrawal J. L. Bruno A. El Abbadi V. Krishnaswamy
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

In the presence of semantic information, serializability is too strong a correctness criterion and unnecessarily restricts concurrency. We use the semantic information of a transaction to provide different atomicity views of the transaction to other transactions. The proposed approach improves concurrency and allows interleavings among transactions which are non-serializable, but which nonetheless preserve the consistency of the database and are acceptable to the users. We develop a graph-based tool whose acyclicity is both a necessary and sufficient condition for the correctness of an execution. Our theory encompasses earlier proposals that incorporate semantic information of transactions. Furthermore it is the first approach that provides an efficient graph based tool for recognizing correct schedules without imposing any restrictions on the application domain. Our approach is widely applicable to many advanced database applications such as systems with long-lived transactions and collaborative environments.

1 Introduction

The traditional approach for transaction management in multi-user database systems is to maintain entire transactions as single atomic units with respect to each other. Such atomicity of transactions is enforced in most commercial database systems by ensuring that the interleaved execution of concurrent transactions remains *serializable* [EGLT76, RSL78, Pap79, BSW79]. Databases are increasingly used in applications, where transactions may be long lived, or where transactions correspond to executions of various users cooperating with each other, e.g., in design databases, CAD/CAM databases, etc. For such applications serializability is found to be too restrictive [BK91].

Researchers, in general, have taken two different approaches to address this problem. Instead of modeling the database as a collection of objects that can only be read or written by transactions, a number of researchers have considered placing more structure on data objects to exploit type specific semantics [Kor83, SS84, Her86, Wei89, BR92]. This approach increases concurrency in the system while remaining within the confines of serializability. The other approach relaxes the absolute atomicity of transactions and uses the explicit semantics of transactions to allow executions in which a transaction may provide different atomicity views to other transactions [Gar83, Lyn83, FÖ89]. For example, consider a transaction T consisting of database operations $o_1 o_2 o_3$. The traditional serializability requirement is that T appears as a single atomic unit $\boxed{o_1 o_2 o_3}$ to other transactions such as T' and T''. However, by using the semantics of transactions, the atomicity specifications of T may be made different with respect to other transactions. The user may specify that T should appear as $\boxed{o_1 o_2}\boxed{o_3}$ to T' but as $\boxed{o_1}\boxed{o_2 o_3}$ to T''. Lynch [Lyn83] introduced this notion and refers to it as *relative atomicity*.

In [Lyn83], several examples are presented to motivate the advantages of relative atomicity for banking applications and computer-aided design environments. In the banking example, customers are grouped into families each of which shares a common set of accounts. The bank may wish to take a complete bank audit of all accounts, while creditors may require a credit audit of specific families. In this case the bank audit should be atomic with respect to all other transactions and vice versa. The relative atomicity specifications for credit audits and customer transactions are much less severe. Finally, customer transactions in the same family can be arbitrarily interleaved. In the computer-aided design example, users are divided into teams of specialized experts. Relative atomicity specifications can now be used to specify different constraints for members within a team and with respect to other teams.

In databases with typed objects several efficient protocols exist for enforcing correct executions of transactions. In contrast, results in the area of relative atomicity are still at a preliminary stage. In particular, there does not exist a well-defined theory for analyzing and arguing the correctness of executions of transactions with relative atomicity specifications. The traditional serializability theory, for example, defines the notion of correct executions which are serial executions and provides a graph-based tool to recognize executions that are "equivalent" to serial executions. If relative atomicity specifications are to be used in practice, we need to develop a theory similar to the traditional serializability theory.

Garcia-Molina [Gar83] pioneered the effort to use the notion of relative atomicity to increase concurrency in database systems. He proposed grouping transactions into compatibility sets, where transactions in one such set may be arbitrarily interleaved, but transactions in different sets observe each other as single atomic units. Clearly, Garcia-Molina's proposal is a special case of transactions with relative atomicity specifications. Lynch [Lyn83] extended Garcia-Molina's specifications from two-level compatibility sets to hierarchically structured interleaving sets, allowing transactions to have varying atomic units relative to each other. Once again, it can be argued that Lynch's hierarchical specifications are restrictive and do not model the general notion of relative atomicity. However, Lynch's proposal is comprehensive in that it not only specifies a notion of correctness but also provides a graph-based tool to efficiently recognize correct executions. Farrag and Özsu [FÖ89] used the notion of breakpoints, which provides maximal freedom for relative atomicity specifications. In addition, they propose a graph which aids in the recognition of executions that are "equivalent" to correct executions. The most serious drawback of Farrag and Özsu's proposal is that the recognition of executions that are equivalent to correct executions has an exponential complexity.

Once the notion of relative atomicity has been defined, the main challenge is to define classes of executions that are larger than the class of correct executions, but that are "equivalent" to the set of correct executions. Extending the definitions from traditional databases to relative atomicity, Farrag and Özsu define an execution to be *correct* if it satisfies the user defined relative atomicity specifications and the class *relatively consistent*, which includes all executions that are *conflict equivalent* to correct executions. Unfortunately, the complexity of recognizing relatively consistent executions is NP-Complete [KB92].

In this paper, we retain the notion of conflict equivalence and reduce the complexity of this problem by modifying the notion of correct executions. This is in contrast to the traditional approach of strengthening the notion of equivalence for reducing the complexity of the problem[1]. Rather than defining a correct execution to be the one that only allows the interleavings specified by the user, our definition takes into account the actual interleavings occurring in the specific execution. Our definition may therefore allow an operation to be interleaved within the relatively atomic unit of another transaction, as long as in the given execution no "dependencies" occur between this operation and the operations of the atomic unit. In the traditional model, this would be analogous to generalizing the class of serial executions by allowing transactions that do not have any "dependencies" to be arbitrarily interleaved. This new definition of correct executions which subsumes all earlier definitions of correct executions, reduces the complexity of recognizing the set of executions that are equivalent to correct executions. In particular, we develop a graph, whose acyclicity is both a necessary and sufficient condition for an execution to be equivalent to a correct execution. Furthermore, when the relative atomicity specifications are restricted to the case where each transaction is an atomic unit, our theory corresponds to the traditional serializability theory.

2 Model

A *database* is modeled as a set of *objects*. The objects in the database can be accessed through atomic *read* and *write* operations. Users interact with the database by invoking *transactions*. A transaction is a sequence of read and write operations that are executed on the objects. A read (write) operation executed by a transaction T_i on object x is denoted as $r_i[x]$ ($w_i[x]$). A *schedule* S over $\mathcal{T} = \{T_1, \ldots, T_n\}$ is an interleaved sequence[2] of all the operations of the transactions in $\mathcal{T}$ such that the operations of transaction T_i appear in the same order in S as they do in T_i, for $i = 1, \ldots, n$. In order to relate schedules over the same set of transactions, the notion of *conflict* between operations is used in concurrency control theory [Pap79, BSW79]. Two operations of different transactions *conflict* if they access the same data object and at least one of them is a write operation. Two schedules are *conflict equivalent* if they both order conflicting operations in the same manner.

Our motivation in this paper is to generalize the transaction model. In particular, if semantics of the applications being modeled by the database is available to the users writing the transactions, they may be able to weaken the requirement of the atomicity of transactions. In the relative atomicity model, a transaction has multiply defined atomicity specifications with respect to every other transaction. Formally, an *atomic unit of* T_i *relative to* T_j is a sequence of operations of T_i such that no operations of T_j are allowed to be executed within this sequence. $Atomicity(T_i, T_j)$ denotes the ordered sequence of atomic units of T_i relative to T_j and $AtomicUnit(k, T_i, T_j)$ denotes the k^{th} atomic unit in $Atomicity(T_i, T_j)$. Figure 1 illustrates an example of three transactions with their relative atomicity specifications. In particular, the rectangular boxes represent the atomic units of each transaction with respect to other transactions. In Figure 1, $Atomicity(T_1, T_2)$ is $\langle\ \boxed{r_1[x] w_1[x]}\ ,\ \boxed{w_1[z], r_1[y]}\ \rangle$, which indicates that if operations of T_2 have to be executed within T_1 then they may only be executed after $w_1[x]$ and before $w_1[z]$. We say that an operation o of a transaction T_j is *interleaved with* an $AtomicUnit(k, T_i, T_j)$ in a schedule S, if there exist

[1] Such an approach is explored in [Kri93].

[2] In general, transactions and schedules are permitted to be partially ordered sequences. In this paper, however, we assume that transactions and schedules are totally ordered sequences for simplicity.

operations o' and o'' of $AtomicUnit(k, T_i, T_j)$ such that o' precedes o and o precedes o'' in S.

Consider a set of transactions $\mathbf{T} = \{T_1, T_2, T_3\}$ where

$$T_1 = r_1[x]w_1[x]w_1[z]r_1[y]$$
$$T_2 = r_2[y]w_2[y]r_2[x]$$
$$T_3 = w_3[x]w_3[y]w_3[z]$$

$Atomicity(T_1, T_2)$: $\boxed{r_1[x]w_1[x]}$ $\boxed{w_1[z]r_1[y]}$
$Atomicity(T_1, T_3)$: $\boxed{r_1[x]w_1[x]}$ $\boxed{w_1[z]}$ $\boxed{r_1[y]}$

$Atomicity(T_2, T_1)$: $\boxed{r_2[y]}$ $\boxed{w_2[y]r_2[x]}$
$Atomicity(T_2, T_3)$: $\boxed{r_2[y]w_2[y]}$ $\boxed{r_2[x]}$

$Atomicity(T_3, T_1)$: $\boxed{w_3[x]w_3[y]}$ $\boxed{w_3[z]}$
$Atomicity(T_3, T_2)$: $\boxed{w_3[x]w_3[y]}$ $\boxed{w_3[z]}$

Figure 1: Relative Atomicity Specifications

The relative atomicity specifications over $\mathcal{T}$ is the set $\{Atomicity(T_i, T_j) | T_i, T_j \in \mathcal{T}\}$. The above specifications could also be specified in terms of transactions types instead of transactions instances as in [Gar83, FÖ89]. However, for simplicity of exposition we will restrict ourselves to relative atomicity in terms of transaction instances [Lyn83]. The relative atomicity specifications can also be specified by using the notion of atomic steps [Gar83] or breakpoints [FÖ89].

A correct execution in the traditional transaction model requires that all operations of a transaction appear as a single atomic unit. We refer to this as *absolute atomicity*. For example, T_1 appears as $\boxed{r_1[x]w_1[x]w_1[z]r_1[y]}$ to all other transactions under absolute atomicity. From the relative atomicity specifications of transactions in Figure 1, in any correct execution it is required that operations of T_2 not be interleaved with $AtomicUnit(k, T_1, T_2)$ for $k = 1, 2$. However, operations of T_2 may be executed between the two atomic units of T_1 relative to T_2. Consider, for example, the following schedule:

$$S_{ra} = r_2[y]r_1[x]w_1[x]w_2[y]r_2[x]w_1[z]w_3[x]w_3[y]r_1[y]w_3[z]$$

Note that even though S_{ra} is not a serial schedule, it is correct with respect to the relative atomicity specifications of the three transactions in Figure 1. For example, operations of T_1 are executed between $r_2[y]$ and $w_2[y]r_2[x]$. In spite of such interleavings, the atomicity of T_2 relative to T_1 is preserved. It can be easily shown that the same holds for the remaining interleavings. Based on the relative atomicity specifications, the correctness of a schedule S over $\mathcal{T}$ can be defined as follows:

Definition 1 S is a *relatively atomic schedule* over $\mathcal{T}$ if for all transactions T_i and T_l no operation of T_i is interleaved with an $AtomicUnit(k, T_l, T_i)$ for any k.

Farrag and Özsu [FÖ89] refer to relatively atomic schedules as correct schedules. Further, they define *relatively consistent* schedules as the schedules that are conflict equivalent to a relatively atomic schedule. Unfortunately, recognizing the class of relatively consistent schedules is NP-complete [KB92].

To reduce the complexity of recognizing relatively consistent schedules, we modify the definition of correct schedules. We motivate our approach by studying the reason for the NP-Completeness with the above definition of correctness. The complexity result arises due to the ambiguity in ordering non-conflicting operations. Consider for example three operations o_1, o_2, and o_3 in a schedule S, where o_1 and o_2 are operations of a transaction T in which o_1 precedes o_2, and o_3 is an operation of a transaction T' $(T \neq T')$, such that $\boxed{o_1 o_2}$ is an atomic unit of T relative to T'. If o_3 conflicts with o_1 and/or o_2, then the order of o_3 relative to $\boxed{o_1 o_2}$ is determined by the conflict. On the other hand, if o_3 does not conflict with o_1 and o_2, there is a choice of ordering o_3 either before o_1 or after o_2 in an equivalent relatively atomic schedule. Depending on the rest of the schedule, one (or both) of these choices may not be possible in any equivalent relatively atomic schedule. In order to verify whether a schedule is relatively consistent, both choices must be considered to determine if there exists an equivalent relatively atomic schedule. Since several such triples may exist for any given schedule, there are potentially an exponential number of choices that must be considered. It might then be necessary to check each of these orderings for membership in the class of relatively atomic schedules, thus making the test for relatively consistent schedules exponential.

In fact, the complexity of the problem occurs only when the given schedule does not impose a particular order between o_3 and o_1 or o_2, but the user's relative atomicity specifications require the exclusion of o_3 from the atomic unit $\boxed{o_1 o_2}$. If a schedule does not impose any order between two operations then the execution of one operation cannot depend on or affect the execution of the other. Otherwise, a sequence of conflicting operations must have been executed, thus enforcing an order. Our approach is therefore to extend the user's relative atomicity specifications and allow o_3 to be executed between o_1 and o_2 when o_3's execution cannot affect or depend on either o_1 or o_2. This unseemly exclusion arises due to the fact that the relative atomicity specifications for the application are given a priori and have to take into account all potential conflicts that might occur in any execution. In a particular execution not all of these potential conflicts occur, and the relative atomicity specifications tend to be conservative. For example, transaction executions that correspond to conditionals, the conflicts that occur can be determined only at execution time. We are therefore motivated to expand the class of relatively atomic schedules to include interleavings of operations which do not have any dependencies between them.

In order to characterize correct schedules in our model, we need to define an additional notion called *depends on* which is derived from conflicts and the internal structure of transactions. We say that o_2 *directly depends on* o_1 if o_1 precedes o_2 in S and either o_1 and o_2 are operations of the same transaction or o_1 conflicts with o_2. The *depends on* relation is the transitive closure of the directly depends on relation. In the schedule S_1 shown in Figure 2, $w_2[y]$ does not conflict with either $w_1[x]$ or $r_1[z]$, but $r_1[z]$ is affected by $w_2[y]$. Since the user's relative atomicity specifications does not allow T_2 in the atomic unit $\boxed{w_1[x] r_1[z]}$, S_1 is not a correct schedule. If the depends on relation is based only on direct conflicts then the schedule S_1 will be considered as a correct schedule. Hence the effects from $w_2[y]$ to $r_1[z]$ should be captured in the depends on relation, so as to rule out S_1 as a correct schedule. Another reason for using the notion of depends on

Consider a set of transactions $\mathbf{T} = \{T_1, T_2, T_3\}$ where

$T_1 = w_1[x]r_1[z]$

$T_2 = w_2[y]$

$T_3 = r_3[y]w_3[z]$

$Atomicity(T_1, T_2)$: $\boxed{w_1[x]r_1[z]}$

$Atomicity(T_1, T_3)$: $\boxed{w_1[x]}$ $\boxed{r_1[z]}$

$Atomicity(T_2, T_1)$: $\boxed{w_2[y]}$

$Atomicity(T_2, T_3)$: $\boxed{w_2[y]}$

$Atomicity(T_3, T_1)$: $\boxed{r_3[y]}$ $\boxed{w_3[z]}$

$Atomicity(T_3, T_2)$: $\boxed{r_3[y]}$ $\boxed{w_3[z]}$

$$S_1 = w_1[x]w_2[y]r_3[y]w_3[z]r_1[z]$$

Figure 2: An example to show that direct conflicts are not sufficient for correctness

relation, which generalizes the notion of conflicts, is to maintain the correspondence between our correctness criterion with the traditional correctness criterion based on serializability. This is discussed below after the formal definition of correct schedules in our model.

For notational convenience, in the rest of the paper, we index transactions and the operations within a transaction. For example, operation o_{ij} is the j^{th} operation of transaction T_i. We now define correct schedules in our model that are referred to as *relatively serial* schedules.

Definition 2 S is a *relatively serial* schedule over $\mathcal{T}$ if for all transactions T_i and T_l, if an operation o_{ij} of T_i is interleaved with an $AtomicUnit(k, T_l, T_i)$ for some k, then o_{ij} does not depend on any operation $o_{lm} \in AtomicUnit(k, T_l, T_i)$ and vice-versa.

Consider the following schedule S_{rs} over $\mathbf{T}$ given in Figure 1:

$$S_{rs} = r_1[x]r_2[y]w_1[x]w_2[y]w_3[x]w_1[z]w_3[y]r_2[x]r_1[y]w_3[z]$$

In S_{rs} operation $r_2[y]$ is interleaved with $AtomicUnit(1, T_1, T_2)$ and $r_2[y]$ does not depend on $r_1[x]$ and $w_1[x]$ does not depend on $r_2[y]$. Similarly for the other interleavings: $w_1[z]$ interleaved with $AtomicUnit(2, T_2, T_1)$ and $AtomicUnit(1, T_3, T_1)$ and $r_2[x]$ interleaved with $AtomicUnit(2, T_1, T_2)$. Hence, S_{rs} is relatively serial.

The notion of relatively serial schedules is analogous to the notion of serial schedules in the serializability theory. In particular, there exist schedules that are not relatively serial but may have the same behavior as some relatively serial schedules. We define a schedule to be *relatively serializable* if it is conflict equivalent to some relatively serial schedule. For example, the following schedule S_1 over $\mathbf{T}$ given in Figure 1:

$$S_2 = r_1[x]r_2[y]w_2[y]w_1[x]w_3[x]r_2[x]w_1[z]w_3[y]r_1[y]w_3[z]$$

is not relatively serial since $w_1[x]$ is interleaved with $AtomicUnit(2, T_2, T_1)$ and $r_2[x]$ depends on $w_1[x]$. However, S_2 is relatively serializable since it is conflict equivalent to the relatively serial schedule S_{rs} given above.

The relative atomicity model is a generalization of the traditional absolute atomicity model. That is, if the specifications are such that each transaction is a single atomic unit with respect to all other transactions, then relative atomicity reduces to absolute atomicity. When restricted to absolute atomicity of transactions, relatively serial schedules are different from serial schedules. Every serial schedule is a relatively serial schedule since each operation of a serial schedule is not interleaved within the atomic unit of any transaction. Relatively serial schedules allow arbitrary interleavings (within atomic units) of operations that do not have any dependencies between them. It is important that the introduction of a new correctness definition should not contradict the correctness criterion in the traditional model. In the following lemma, we show that the definition of relatively serial schedules is such that, under absolute atomicity, the same correctness as that in the traditional model is still maintained.

Lemma 1 *When relative atomicity is restricted to the traditional absolute atomicity model, any relatively serial schedule is conflict equivalent to some serial schedule.*

Proof: Consider a relatively serial schedule S_{rs} on $\mathcal{T}$. If S_{rs} is serial then the lemma is trivially satisfied. If S_{rs} is not serial, assume for contradiction that there does not exist any serial schedule that is conflict equivalent to S_{rs}. Consider the serialization graph $SG(S_{rs})$ for S_{rs} [Pap79, BSW79]. $SG(S_{rs})$ has transactions in $\mathcal{T}$ as its nodes and $T_i \longrightarrow T_k$ is an edge in $SG(S_{rs})$ if an operation of T_i conflicts and precedes an operation of T_k in S_{rs}. Since S_{rs} is not conflict serializable, $SG(S_{rs})$ has a cycle. We show that for each edge from $T_i \longrightarrow T_k$ in $SG(S_{rs})$, o_{i1} precedes o_{k1} in S_{rs}. Consider an edge $T_i \longrightarrow T_k$ in $SG(S_{rs})$. This implies that there exist operations, say, o_{ij} and o_{kl}, such that o_{ij} conflicts and precedes o_{kl} in S_{rs}. Since S_{rs} is relatively serial, o_{ij} precedes the atomic unit of T_k. Thus o_{ij} precedes o_{k1} in S_{rs} and consequently o_{i1} precedes o_{k1} in S_{rs}. This implies that for any cycle in $SG(S_{rs})$, the first operation of every transaction in the cycle precedes itself in S_{rs}, a contradiction. Hence the lemma. $\Box$

In the absolute atomicity model, every serial schedule is a relatively serial schedule. From the definition of relatively serializable schedules and Lemma 1, it is clear that under absolute atomicity, any relatively serializable schedule is equivalent to a serial schedule. Thus, the set of relatively serializable schedules is exactly the same as the set of conflict serializable schedules [Pap79, BSW79] under absolute atomicity. Hence, our model retains the standard notion of correctness when the relative atomicity specifications are restricted to specify the traditional transaction model.

3 Efficient Testing of Relatively Serializable Schedules

In this section we develop a graph that can be used to determine whether a given schedule is relatively serializable. Our approach is analogous to the traditional serializability theory in which a serialization graph is used to determine whether a given schedule is conflict serializable. We define two complementary notions of pushing forward or pulling backward an operation. Let o_{ij} belong to $AtomicUnit(m, T_i, T_k)$. We define

$PushForward(o_{ij}, T_k)$ to be operation o_{ip} where o_{ip} is the last operation of $AtomicUnit(m, T_i, T_k)$. Analogously, we define $PullBackward(o_{ij}, T_k)$ to be operation o_{iq} where o_{iq} is the first operation of $AtomicUnit(m, T_i, T_k)$. For example in Figure 1, $PushForward(r_1[x], T_2)$ is $w_1[x]$ and $PullBackward(r_1[y], T_2)$ is $w_1[z]$. The motivation for the above two notions is as follows. Let an operation o_{kl} of transaction T_k be interleaved between operations o_{ij} and o_{ij+1} of another transaction T_i in some schedule S. Furthermore, assume T_k is not allowed to interleave between o_{ij} and o_{ij+1} as per the relative atomicity specifications. In that case, S can be "corrected" if o_{kl} can be pulled backward before $PullBackward(o_{ij+1}, T_k)$ or pushed forward after $PushForward(o_{ij}, T_k)$. However, since our definition of relatively serial schedules allows certain operations to be interleaved in an atomic unit (i.e., operations that do not have the depends on relation with operations in an atomic unit), the pushing and pulling needs to be performed conditionally. We formalize these arguments in the following definition of the *relative serialization graph*.

Definition 3 The *relative serialization graph* of a schedule S over $\mathcal{T}$, denoted $RSG(S) = (V, E)$, is a directed graph whose vertices V are the set of operations of the transactions in $\mathcal{T}$ and whose arcs E are as follows:

1. *Internal Arcs* (*I-arcs*). E contains all the internal arcs of the form $o_{ij} \longrightarrow o_{ij+1}$ where o_{ij} and o_{ij+1} are consecutive operations in T_i for all $T_i \in \mathcal{T}$.
2. *Dependency Arcs* (*D-arcs*). E contains all the dependency arcs of the form $o_{ij} \longrightarrow o_{kl}$ such that $i \neq k$ and o_{kl} depends on o_{ij} in S. Note that these arcs also capture conflicts.
3. *Push Forward Arcs* (*F-arc*). For each *D-arc* $o_{ij} \longrightarrow o_{kl}$ we add $PushForward(o_{ij}, T_k) \longrightarrow o_{kl}$ in E.
4. *Pull Backward Arcs* (*B-arc*). For each *D-arc* $o_{kl} \longrightarrow o_{ij}$ we add $o_{kl} \longrightarrow PullBackward(o_{ij}, T_k)$ in E.

Lynch [Lyn83] as well as Farrag and Özsu [FÖ89] use the notion of pushing forward an operation out of an atomic unit. However, neither of them employed the notion of pulling backward an operation out of an atomic unit. In Figure 3, we provide an example of the relative serialization graph resulting from the following schedule S_2 over $\mathbf{T}$:

$$S_2 = w_1[x]r_2[x]r_3[z]w_2[y]r_3[y]r_1[z]$$

For example, since $\boxed{w_1[x]r_1[z]}$ is atomic with respect to T_2 and since $r_2[x]$ depends on $w_1[x]$, $RSG(S_2)$ contains the *F-arc* from $r_1[z]$ to $r_2[x]$. Since $\boxed{r_3[z]r_3[y]}$ is atomic relative to T_2 and $r_3[y]$ depends on $w_2[y]$, $RSG(S_2)$ contains the *B-arc* from $w_2[y]$ to $r_3[z]$.

We now prove that acyclicity of $RSG(S)$ is both a necessary and sufficient condition for schedule S to be relatively serializable. In the following, we say that an arc $o \longrightarrow o'$ in $RSG(S)$ is *consistent* with S if o precedes o' in S. We start by establishing that the relative serialization graph of a relatively serial schedule is acyclic.

Lemma 2 *If S is a relatively serial schedule over $\mathcal{T}$ then $RSG(S)$ is acyclic.*

Proof: We will show that every arc of $RSG(S)$ is consistent with S. It is easy to see that the *I-arcs* and *D-arcs* in $RSG(S)$ are consistent with S. Consider an *F-arc*, $o_{ip} \longrightarrow o_{kl}$ in $RSG(S)$. This implies that there is a *D-arc* from $o_{ij} \longrightarrow o_{kl}$, $j \leq p$ where o_{ij} and o_{ip} are in the same atomic unit, say $AtomicUnit(m, T_i, T_k)$, and o_{ip} is the last operation in $AtomicUnit(m, T_i, T_k)$. Since o_{kl} depends on o_{ij}, and S is relatively serial, o_{kl} must appear after $AtomicUnit(m, T_i, T_k)$ in S. Thus the *F-arc* $o_{ip} \longrightarrow o_{kl}$ in $RSG(S)$ is consistent with S. Similarly, it can be shown that any *B-arc* in $RSG(S)$ is consistent with S. Since all arcs in $RSG(S)$ are consistent with S and S is a total order, $RSG(S)$ is acyclic. □

Consider a set of transactions $\mathbf{T} = \{T_1, T_2, T_3\}$ where

$T_1 = w_1[x]r_1[z]$

$T_2 = r_2[x]w_2[y]$

$T_3 = r_3[z]r_3[y]$

$Atomicity(T_1, T_2)$: $\boxed{w_1[x]r_1[z]}$

$Atomicity(T_1, T_3)$: $\boxed{w_1[x]}$ $\boxed{r_1[z]}$

$Atomicity(T_2, T_1)$: $\boxed{r_2[x]}$ $\boxed{w_2[y]}$

$Atomicity(T_2, T_3)$: $\boxed{r_2[x]}$ $\boxed{w_2[y]}$

$Atomicity(T_3, T_1)$: $\boxed{r_3[z]}$ $\boxed{r_3[y]}$

$Atomicity(T_3, T_2)$: $\boxed{r_3[z]r_3[y]}$

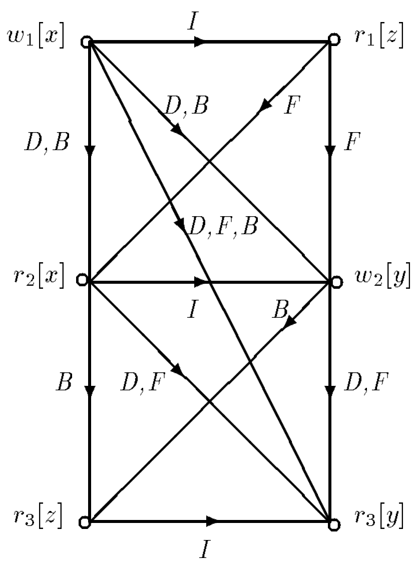

Figure 3: An example of a relative serialization graph

Theorem 1 *A schedule S over $\mathcal{T}$ is relatively serializable if and only if $RSG(S)$ is acyclic.*

Proof: Assume S is relatively serializable. Then there exists a relatively serial schedule S_{rs} over $\mathcal{T}$ which is conflict equivalent to S. The *I-arcs* are obviously the same in $RSG(S)$ and $RSG(S_{rs})$. The *D-arcs* of $RSG(S)$ and $RSG(S_{rs})$ are identical because of the conflict equivalence of S and S_{rs}. Since the *F-arcs* and *B-arcs* depend only on the *I-arcs*, *D-arcs*, and the relative atomicity specifications for $\mathcal{T}$, $RSG(S)$ and $RSG(S_{rs})$ are identical. It follows from Lemma 2 that $RSG(S)$ is acyclic.

Assume that $RSG(S)$ is acyclic. Let S_{rs} be a schedule obtained by topologically sorting $RSG(S)$. Clearly, S and S_{rs} are conflict equivalent and therefore $RSG(S_{rs})$ is the same as $RSG(S)$. It remains to show that S_{rs} is relatively serial. Assume for contradiction that there exists an operation o_{ij} interleaved with $AtomicUnit(m, T_k, T_i)$ which contains o_{kl} such that o_{ij} depends on o_{kl} or o_{kl} depends on o_{ij} in S_{rs}. Let o_{kp} be the last operation of $AtomicUnit(m, T_k, T_i)$ and let o_{kq} be the first operation of $AtomicUnit(m, T_k, T_i)$. If o_{ij} depends on o_{kl} in S_{rs} then there is an *F-arc* $o_{kp} \longrightarrow o_{ij}$ in $RSG(S)$, hence in any topological sort of

$RSG(S)$, o_{kp} precedes o_{ij}. However, o_{ij} precedes o_{kp} in S_{rs} since o_{ij} is interleaved in $AtomicUnit(m, T_k, T_i)$, a contradiction. If o_{kl} depends on o_{ij} in S_{rs} then there is a *B-arc* $o_{ij} \longrightarrow o_{kq}$ in $RSG(S)$, hence in any topological sort of $RSG(S)$, o_{ij} precedes o_{kq}. Again we obtain a contradiction since o_{kq} precedes o_{ij} in S_{rs} by our assumption. Therefore S_{rs} is relatively serial. □

The relative serialization graph provides an efficient (polynomial) method for recognizing relatively serializable schedules. This graph can be used as the basis for a concurrency control protocol similar to serialization graph testing [Bad79, Cas81]. We are currently investigating locking protocols for ensuring relative serializability of transactions with relative atomicity.

4 Related Work

The relative atomicity model is a generalization of the traditional absolute atomicity model. In particular, our model retains the standard notion of correctness based on on conflict serializability [Pap79, BSW79] when the relative atomicity specifications are restricted to specify the traditional transaction model. Although the proposed model is useful in environments where transaction semantics is available, it may as well be used in the standard database systems without jeopardizing correctness.

One of the earliest proposals to depart from absolute atomicity of transactions was proposed by Garcia-Molina [Gar83]. However, his approach for relaxing atomicity of transactions is a special case of relative atomicity. Although Lynch [Lyn83] proposed relative atomicity, her model is restricted to hierarchical relative atomicity specifications called *multilevel atomicity*. Lynch has proposed a graph-based tool for efficiently recognizing schedules that are equivalent to multilevel atomic schedules. From the users point of view, however, multilevel atomicity imposes several constraints in specifying interleavings thus reducing its applicability. Relative atomicity, on the other hand, imposes no constraints on the user specifications. It is easy to construct examples that can be specified using relative atomicity but cannot be specified using multilevel atomicity.

We now compare the set of relatively consistent schedules [FÖ89] and the set of relatively serializable schedules under relative atomicity. Since relatively consistent schedules are conflict equivalent to relatively atomic [FÖ89] schedules and the set of relatively atomic schedules is a subset of the set of relatively serial schedules, it follows that relatively consistent schedules are also relatively serializable. The proper containment of the set of relatively consistent schedules in the set of relatively serializable schedules is demonstrated by the example in Figure 4. The schedule S given in Figure 4 is a relatively serial schedule. However, S is not conflict equivalent to any relatively atomic schedule, since the operations of T_1 cannot be moved out of the atomic unit of T_3 as seen by T_1. In fact operations $w_1[x]$ and $w_1[y]$ cannot be rearranged to obtain equivalent relatively atomic schedule since T_4 and T_2 do not permit T_1 in their respective atomic units. The relationships among the classes described above is given in Figure 5. The inclusions of the sets follow from the definitions. The example given in Figure 4 shows that the set of relatively serializable schedules properly contains the set of relatively consistent schedules defined by Farrag and Özsu [FÖ89].

There have been other proposals to weaken the atomicity of transactions for improving concurrency. However, these approaches remain within the confines of traditional serializability. Their primary goal is to relax the two phase restriction of strict two phase locking. Wolfson [Wol86, Wol87] uses preanalysis of read

Consider a set of transactions $\mathbf{T} = \{T_1, T_2, T_3, T_4\}$ where

$T_1 = w_1[x]w_1[y]$

$T_2 = w_2[z]w_2[y]$

$T_3 = w_3[t]w_3[z]$

$T_4 = w_4[x]w_4[t]$

$Atomicity(T_1, T_2)$: $\boxed{w_1[x]w_1[y]}$

$Atomicity(T_1, T_3)$: $\boxed{w_1[x]w_1[y]}$

$Atomicity(T_1, T_4)$: $\boxed{w_1[x]w_1[y]}$

$Atomicity(T_2, T_1)$: $\boxed{w_2[z]w_2[y]}$

$Atomicity(T_2, T_3)$: $\boxed{w_2[z]w_2[y]}$

$Atomicity(T_2, T_4)$: $\boxed{w_2[z]}\ \boxed{w_2[y]}$

$Atomicity(T_3, T_1)$: $\boxed{w_3[t]w_3[z]}$

$Atomicity(T_3, T_2)$: $\boxed{w_3[t]}\ \boxed{w_3[z]}$

$Atomicity(T_3, T_4)$: $\boxed{w_3[t]}\ \boxed{w_3[z]}$

$Atomicity(T_4, T_1)$: $\boxed{w_4[x]w_4[t]}$

$Atomicity(T_4, T_2)$: $\boxed{w_4[x]}\ \boxed{w_4[t]}$

$Atomicity(T_4, T_3)$: $\boxed{w_4[x]}\ \boxed{w_4[t]}$

$$S = w_4[x]w_3[t]w_4[t]w_1[x]w_1[y]w_2[z]w_2[y]w_3[z]$$

Figure 4: A relatively serial schedule that is not relatively consistent

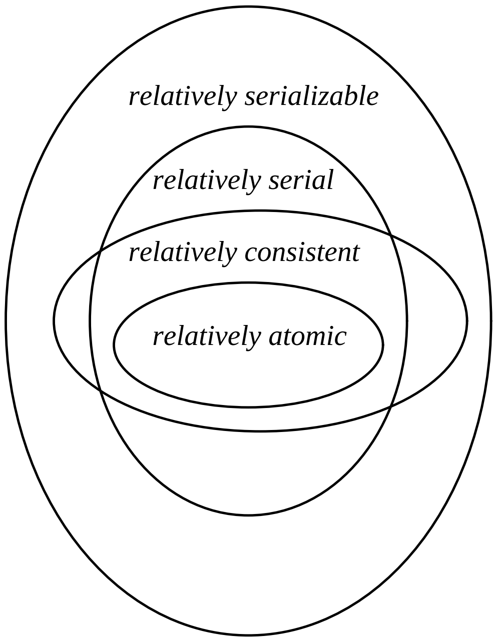

Figure 5: Relationships among the different correctness classes

and write sets of transactions to eliminate the two phase locking rule. Shasha et al. [SSV92] have proposed a chopping graph to refine user transactions such that only the smaller units of the transactions instead of the entire one need to be executed using strict two phase locking.

5 Discussion

In this paper we have developed a theory for relaxing the atomicity of transactions to increase concurrency in database systems. Our approach is general in that it imposes minimal restrictions on the users to specify relative atomicity of transactions. We have shown that the set of relatively serializable schedules defined in this paper is larger than all previous proposals. In spite of being more permissive, this class can be efficiently recognized by testing for the acyclicity of a directed graph. In fact, acyclicity of the graph is both a necessary and sufficient condition for correctness.

We draw an analogy from the historical development of serializability theory to provide some insight into the significance of our approach. In the traditional transaction model, view serializability represented the intuitive correctness criterion based on the user's view of the database. This resulted in a class of schedules which were found to be intractable. Therefore a restrictive class containing conflict serializable schedules was accepted as the set of correct schedules. In the relative atomicity model the equivalence to relatively atomic schedules is the intuitive correctness criterion based on the user's relative atomicity specifications. However in spite of using conflict equivalence this correctness criterion results in a class of schedules which is NP-complete. As we have addressed earlier the relative atomicity specifications tend to be conservative. Hence we relaxed the class of correct schedules to allow for interleavings which can be determined to be correct only at execution time. The theory we have developed is based on a generalization of the notion of conflicts and it provides us with a class which can be efficiently recognized with a graph based tool. Unlike in the traditional model, we have been able to make the problem tractable without having to compromise on the size of the new class. In fact, the class of relatively serializable schedules is larger than the class of relatively consistent schedules. The next step, in traditional databases was the development of more efficient locking based protocols such as the two phase locking protocol [EGLT76] which recognize subsets of the set of conflict serializable schedules. We are currently developing such efficient, lock based protocols for recognizing relatively serializable executions.

We conclude by emphasizing the wide applicability of relative atomicity to many advance database applications. In particular, we envision relative atomicity to be especially useful for systems with long lived transactions as well as several advanced collaborative database environments. In fact, for long live transactions, Salem, Garcia-Molina and Alonso [SGMA87] proposed the use of altruistic locking where a transaction is allowed to explicitly release a lock early so as to allow other transactions to observe it results. In [SGMA87] it is argued and experimentally shown that such locks provide improved performance for long-lived transactions. Relative atomicity can be viewed as a natural generalization of this approach where different degrees of atomicity are allowed with respect to different transactions. A long-lived transactions does not need to be atomic for its entire duration with respect to all other transactions. Rather, different atomic units may be allowed, thus providing more flexibility and concurrency in the system. Finally, in the case of collaborative advanced database applications, such as CAD/CAM design environments, users are often partitioned naturally into groups with various information flow and dependency constraints. Relative atomicity specifications can be easily used to capture such interdependencies. For example, within each group any interleavings may be allowed while different atomicity units can be specified among the different groups depending on the degree of collaboration. Most existing proposals for collaboration are ad-hoc and lack a clear theoretical foundation. We believe that our model provides such a foundation. Furthermore, the main hurdle, in our opinion, for using relative atomicity technique in such environments was due to their computational complexity. We believe that our proposal represents a significant first step towards alleviating these problems, and will make such interactions easy and efficiently manageable.

References

[Bad79] D. Z. Badal. Correctness of Concurrency Control and Implications in Distributed Databases. In *IEEE Proceedings of COMPSAC Conference*, pages 588–593, November 1979.

[BK91] N. S. Barghouti and G. E. Kaiser. Concurrency Control in Advanced Database Applications. *ACM Computing Surveys*, 23(3):269–318, September 1991.

[BR92] B. R. Badrinath and K. Ramamritham. Semantics-Based Concurrency Control: Beyond Commutativity. *ACM Transactions on Database Systems*, 17(1):163–199, March 1992.

[BSW79] P. A. Bernstein, D. W. Shipman, and W. S. Wong. Formal Aspects of Serializability in Database Concurrency Control. *IEEE Transactions on Software Engineering*, 5(5):203–216, May 1979.

[Cas81] M. A. Casanova. *The Concurrency Control Problem of Database Systems*, volume 116 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1981.

[EGLT76] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The Notions of Consistency and Predicate Locks in a Database System. *Communications of the ACM*, 19(11):624–633, November 1976.

[FÖ89] A. A. Farrag and M. T. Özsu. Using Semantic Knowledge of Transactions to Increase Concurrency. *ACM Transactions on Database Systems*, 14(4):503–525, December 1989.

[Gar83] H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Transactions on Database Systems*, 8(2):186–213, June 1983.

[Her86] M. Herlihy. A Quorum-Consensus Replication Method for Abstract Data Types. *ACM Transactions on Computer Systems*, 4(1):32–53, February 1986.

[KB92] V. Krishnaswamy and J. Bruno. On the Complexity of Concurrency Control using Semantic Information. Technical Report TRCS 92-21, Department of Computer Science, University of California, Santa Barbara, CA 93106, 1992.

[Kor83] H. F. Korth. Locking Primitives in a Database System. *Journal of the ACM*, 30(1):55–79, January 1983.

[Kri93] V. Krishnaswamy. *Semantics based Concurrency Control*. PhD thesis, Department of Computer Science, University of California, Santa Barbara, 1993. In preparation.

[Lyn83] N. A. Lynch. Multilevel Atomicity – A New Correctness Criterion for Database Concurrency Control. *ACM Transactions on Database Systems*, 8(4):485–502, December 1983.

[Pap79] C. H. Papadimitriou. The Serializability of Concurrent Database Updates. *Journal of the ACM*, 26(4):631–653, October 1979.

[RSL78] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis. System Level Concurrency Control for Distributed Database Systems. *ACM Transactions on Database Systems*, 3(2):178–198, June 1978.

[SGMA87] K. M. Salem, H. Garcia-Molina, and R. Alonso. Altruistic Locking: A Strategy for Coping with Long-Lived Transactions. In D. Gawlick, M. Haynie, and A. Reuter, editors, *Proceedings of the Workshop on High Performance Transaction Processing*, volume 359 of *Lecture Notes in Computer Science*, pages 175–199. Springer-Verlag, 1987.

[SS84] P. M. Schwarz and A. Z. Spector. Synchronizing Shared Abstract Types. *ACM Transactions on Computer Systems*, 2(3):223–250, August 1984.

[SSV92] D. Shasha, E. Simon, and P. Valduriez. Simple Rational Guidance for Chopping Up Transactions. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 298–307, June 1992.

[Wei89] W. E. Weihl. Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types. *ACM Transactions on Programming Languages and Systems*, 11(2):249–283, April 1989.

[Wol86] O. Wolfson. An Algorithm for Early Unlocking of Entities in Database Transactions. *Journal of Algorithms*, 7(1):146–156, 1986.

[Wol87] O. Wolfson. The Virtues of Locking by Symbolic Names. *Journal of Algorithms*, 8:536–556, 1987.