Algebraic Specification of Serializability for Partitioned Transactions

Walter Hussak and John Keane

Abstract—The usual correctness condition for a schedule of concurrent database transactions is some form of serializability of the transactions. For general forms, the problem of deciding whether a schedule is serializable is NP-complete. In those cases other approaches to proving correctness, using proof rules that allow the steps of the proof of serializability to be guided manually, are desirable. Such an approach is possible in the case of conflict serializability which is proved algebraically by deriving serial schedules using commutativity of non-conflicting operations. However, conflict serializability can be an unnecessarily strong form of serializability restricting concurrency and thereby reducing performance. In practice, weaker, more general, forms of serializability for extended models of transactions are used. Currently, there are no known methods using proof rules for proving those general forms of serializability. In this paper, we define serializability for an extended model of partitioned transactions, which we show to be as expressive as serializability for general partitioned transactions. An algebraic method for proving general serializability is obtained by giving an initial-algebra specification of serializable schedules of concurrent transactions in the model. This demonstrates that it is possible to conduct algebraic proofs of correctness of concurrent transactions in general cases.

Keywords—Algebraic Specification, Partitioned Transactions, Serializability.

I. INTRODUCTION

The standard database transaction model [4] has proved to be inadequate when applied practically, for example to transactions of long duration. Requiring serializability of such transactions results in long duration waits and abortion of long transactions. A proposed solution to overcome these problems has been to partition transactions. Models of partitioned transactions and suggested application areas have been given in [16], [11] and [5]. These allow an increase in concurrency for executing transactions whilst maintaining database consistency based on data consistency conditions weaker than standard serializability. However, data consistency conditions are useful only if there are suitable methods for proving them. In the standard model of transactions, this means proving serializability. The proof techniques are algorithms based on dependency graphs [12] and [17]. These algorithms are NP-complete and for this reason stronger forms of serializability that can be proved in polynomial time, notably ‘conflict’ serializability [12], have been suggested. Interestingly, conflict serializability can also be characterized algebraically - a schedule is serializable if and only if it can be transformed into a serial schedule by commuting non-conflicting operations (i.e. two reads or two operations accessing different data items). Such

I. INTRODUCTION

The standard database transaction model [4] has proved to be inadequate when applied practically, for example to transactions of long duration. Requiring serializability of such transactions results in long duration waits and abortion of long transactions. A proposed solution to overcome these problems has been to partition transactions. Models of partitioned transactions and suggested application areas have been given in [16], [11] and [5]. These allow an increase in concurrency for executing transactions whilst maintaining database consistency based on data consistency conditions weaker than standard serializability. However, data consistency conditions are useful only if there are suitable methods for proving them. In the standard model of transactions, this means proving serializability. The proof techniques are algorithms based on dependency graphs [12] and [17]. These algorithms are NP-complete and for this reason stronger forms of serializability that can be proved in polynomial time, notably ‘conflict’ serializability [12], have been suggested. Interestingly, conflict serializability can also be characterized algebraically - a schedule is serializable if and only if it can be transformed into a serial schedule by commuting non-conflicting operations (i.e. two reads or two operations accessing different data items). Such

I. INTRODUCTION

The standard database transaction model [4] has proved to be inadequate when applied practically, for example to transactions of long duration. Requiring serializability of such transactions results in long duration waits and abortion of long transactions. A proposed solution to overcome these problems has been to partition transactions. Models of partitioned transactions and suggested application areas have been given in [16], [11] and [5]. These allow an increase in concurrency for executing transactions whilst maintaining database consistency based on data consistency conditions weaker than standard serializability. However, data consistency conditions are useful only if there are suitable methods for proving them. In the standard model of transactions, this means proving serializability. The proof techniques are algorithms based on dependency graphs [12] and [17]. These algorithms are NP-complete and for this reason stronger forms of serializability that can be proved in polynomial time, notably ‘conflict’ serializability [12], have been suggested. Interestingly, conflict serializability can also be characterized algebraically - a schedule is serializable if and only if it can be transformed into a serial schedule by commuting non-conflicting operations (i.e. two reads or two operations accessing different data items). Such

I. INTRODUCTION

The standard database transaction model [4] has proved to be inadequate when applied practically, for example to transactions of long duration. Requiring serializability of such transactions results in long duration waits and abortion of long transactions. A proposed solution to overcome these problems has been to partition transactions. Models of partitioned transactions and suggested application areas have been given in [16], [11] and [5]. These allow an increase in concurrency for executing transactions whilst maintaining database consistency based on data consistency conditions weaker than standard serializability. However, data consistency conditions are useful only if there are suitable methods for proving them. In the standard model of transactions, this means proving serializability. The proof techniques are algorithms based on dependency graphs [12] and [17]. These algorithms are NP-complete and for this reason stronger forms of serializability that can be proved in polynomial time, notably ‘conflict’ serializability [12], have been suggested. Interestingly, conflict serializability can also be characterized algebraically - a schedule is serializable if and only if it can be transformed into a serial schedule by commuting non-conflicting operations (i.e. two reads or two operations accessing different data items). Such

I. INTRODUCTION

The standard database transaction model [4] has proved to be inadequate when applied practically, for example to transactions of long duration. Requiring serializability of such transactions results in long duration waits and abortion of long transactions. A proposed solution to overcome these problems has been to partition transactions. Models of partitioned transactions and suggested application areas have been given in [16], [11] and [5]. These allow an increase in concurrency for executing transactions whilst maintaining database consistency based on data consistency conditions weaker than standard serializability. However, data consistency conditions are useful only if there are suitable methods for proving them. In the standard model of transactions, this means proving serializability. The proof techniques are algorithms based on dependency graphs [12] and [17]. These algorithms are NP-complete and for this reason stronger forms of serializability that can be proved in polynomial time, notably ‘conflict’ serializability [12], have been suggested. Interestingly, conflict serializability can also be characterized algebraically - a schedule is serializable if and only if it can be transformed into a serial schedule by commuting non-conflicting operations (i.e. two reads or two operations accessing different data items). Such

I. INTRODUCTION

The standard database transaction model [4] has proved to be inadequate when applied practically, for example to transactions of long duration. Requiring serializability of such transactions results in long duration waits and abortion of long transactions. A proposed solution to overcome these problems has been to partition transactions. Models of partitioned transactions and suggested application areas have been given in [16], [11] and [5]. These allow an increase in concurrency for executing transactions whilst maintaining database consistency based on data consistency conditions weaker than standard serializability. However, data consistency conditions are useful only if there are suitable methods for proving them. In the standard model of transactions, this means proving serializability. The proof techniques are algorithms based on dependency graphs [12] and [17]. These algorithms are NP-complete and for this reason stronger forms of serializability that can be proved in polynomial time, notably ‘conflict’ serializability [12], have been suggested. Interestingly, conflict serializability can also be characterized algebraically - a schedule is serializable if and only if it can be transformed into a serial schedule by commuting non-conflicting operations (i.e. two reads or two operations accessing different data items). Such
version. If \( T = \{ T_1, \ldots, T_n \} \) is a transaction system then \( h \) is a history of \( T \) if the steps of \( h \) are the steps of all the transactions in \( T \) placed in some order such that steps that are ordered in each \( T_i \) preserve that order in \( h \), that is
\[
s_1 <_i s_2 \implies s_1 <_h s_2
\]
A history of \( T \) represents an execution of the transactions in \( T \) in an interleaved fashion.

As an example, suppose that a transaction \( T_i \) reads an integer data item \( a \), updates \( a \), reads an integer data item \( b \), and updates \( b \). Then, \( T_i \) is seen as just a succession of formal reads and writes to \( a \) and \( b \) thus:
\[
r(1, a)w(1, a)r(1, b)w(1, b)
\]
without further information about the nature of \( a \) or \( b \) or the read or write steps. If there are two such transactions \( T_1 \) and \( T_2 \), then the following sequence of steps is a history:
\[
r(1, a)w(1, a)r(2, a)w(2, a)r(2, b)w(2, b)r(1, b)w(1, b)
\]
B. Partitions and tiers
Let \( T_P = \{ T_1, \ldots, T_n \} \) be a transaction system. Then a partition of \( T \) is a collection of partitions of the individual \( T_i \)’s
\[
\mathcal{P} = \{ \mathcal{P}_1, \ldots, \mathcal{P}_n \}
\]
where \( \mathcal{P}_i = \{ T_{i1}, \ldots, T_{ip_i} \} \) is a partition of the set of steps of \( T_i \) (1 ≤ \( i \) ≤ \( n \)). If we wish to indicate that a step belongs to \( T_{ij} \) in the partition \( \mathcal{P}_i \), we shall add an extra (first) argument as in:
\[
r(j, i, x) \text{ or } w(j, i, x)
\]
Each member \( T_{ij} \) of a partition \( \mathcal{P}_i \) is called a \( \mathcal{P}_i \)-segment (or just segment). If, in addition, for each \( i \), \( q \) and \( r \) (and \( q \) and \( r \) distinct) it is impossible to have steps \( s_1, s_2, s_3, \) and \( s_4 \) such that
\[
s_1, s_2 \in T_{iq}, \ s_3, s_4 \in T_{ir}
\]
and
\[
s_1 <_i s_3 \text{ and } s_4 <_i s_2
\]
then \( \mathcal{P} \) is a tiered partition. Each segment is then called a tier, and a transaction partitioned into tiers will be called a tiered transaction. We shall use the tiered transaction model as our model of partitioned transactions. Tiers have a high intuitive appeal. By condition (1), a transaction that is partitioned into tiers can be pictured as being constructed of a succession of tiers placed end to end without overlap. This gives the tiered partitioned model simple algebraic properties. An example of partitioned transactions will be given in III.D below.

C. History equivalence and tiered-serializability
Two histories are ‘equivalent’ iff they have the same set of read and write steps, and every read step ‘sees’ (i.e. reads) the same value in both and also the final database state is the same for both. In detail, let \( h \) be any history \( s_1 \ldots s_K \) (not necessarily of any particular transaction system). A read step
\[
\text{for every read step } s_k \text{ and write step } s_l \text{ and, furthermore, for each variable } x, h \text{ and } h' \text{ have the same last write steps.}
\]
A history of a transaction system \( T \) is serial iff given (distinct) transactions \( T_{i_1}, T_{i_2} \) in \( T \) either
\[
s_1 <_{i_1} s_2 \text{ for all } s_1 \in T_{i_1}, s_2 \in T_{i_2}
\]
and serializable iff it is equivalent to a serial history. If the transactions are partitioned by a tiered partition \( \mathcal{P}_i \), then a history of \( T \) is \( \mathcal{P}_i \)-serial or just tiered-serial (with the understanding that the tiers are as in \( \mathcal{P}_i \)) iff given distinct transactions \( T_{i_1} \) and \( T_{i_2} \) in \( T \), and tiers \( T_{i_1}\)j, \( T_{i_2}\)j, either
\[
s_1 <_{i_1} s_2 \text{ for all } s_1 \in T_{i_1}\)j, s_2 \in T_{i_2}\)j or
\[
s_2 <_{i_2} s_1 \text{ for all } s_1 \in T_{i_1}\)j, s_2 \in T_{i_2}\)j
\]
Clearly, a serial history is tiered-serial. A history \( h \) is \( \mathcal{P}_i \)-serializable or just tiered-serializable iff it is equivalent to a tiered-serial history.

D. An example of tiered-serializability
The reason for tiered serializability is seen from the following example. Suppose that a travel database has data items \( x \) and \( y \) corresponding to booking information for flights between destinations \( A \) and \( B \) and flights between destinations \( B \) and \( C \) respectively. Further, suppose that transactions \( T_i \) arrange journeys between destinations \( A \) and \( C \), by first obtaining flight booking information and booking flights between \( A \) and \( B \) and then obtaining flight information and booking connecting flights between \( B \) and \( C \), so that each \( T_i \) comprises the following succession of read and write steps:
\[
r(1, x)w(1, x)r(1, y)w(1, y)
\]
Consider the following history for the concurrent execution of three such transactions \( T_1, T_2 \) and \( T_3 \):
\[
r(1, x)w(1, x)r(2, x)w(2, x)r(2, y)w(2, y)
\]
\[
r(1, y)r(3, x)w(3, y)r(3, x)w(3, y)
\]
In the history (2), \( T_1 \) and \( T_2 \) are not serialized because \( T_1 \) books its first flight and then \( T_2 \) books both its first and
Connecting flight before $T_1$ manages to book its connecting flight - if all of $T_1$ had been executed before $T_2$, $T_1$ would, in general, have had more connecting flights to choose from and may have booked those differently; if all of $T_2$ had been executed before $T_1$, $T_2$ would, in general, have had more first flights to choose from and may have booked those differently. However, the history (2) still books first and connecting flights consistently and allowing it increases concurrency. Here, consistency means tiered-serializability where the accessed to first flight and connecting flight information respectively constitute the tiers, i.e. each $T_i$ is the tiered transaction:

$$r(1, i, x)w(1, i, x)r(2, i, y)w(2, i, y)$$

and (2) becomes the tiered-serializable history:

$$r(1, 1, x)w(1, 1, x)r(1, 2, x)w(1, 2, x)r(2, 2, y)w(2, 2, y)$$

$$r(2, 1, y)r(1, 3, x)w(2, 1, y)r(1, 3, x)r(2, 3, y)w(2, 3, y)$$

as swapping the places of $r(1, 3, x)$ and $w(2, 1, y)$ does not affect the $\text{sees}$ function. Notice that serializability of at least the tiers is needed as, for example, the history

$$r(1, 1, x)r(1, 2, x)w(1, 1, x)r(1, 2, x)r(2, 2, y)w(2, 2, y)$$

and

$$r(2, 1, y)r(1, 3, x)w(2, 1, y)r(1, 3, x)r(2, 3, y)w(2, 3, y)$$

can result in an inconsistent database state, as flights booked by transaction $T_1$ may be overwritten by those booked by $T_2$. Also, notice that tiers cannot be treated as separate transactions because of interdependencies between tiers belonging to a common transaction - in the example just given, the particular connecting flights booked depend on the particular first flights booked for reasons such as connections being able to be made.

### III. Expressiveness

In Section II above, we defined $P_i$-serializability for tiered partitions $P_i$. A similar definition of $P$-serializability (see below) can be given for arbitrary partitions $P$ that do not satisfy condition (1), and corresponds to serializability for general models of partitioned transactions such as that given in [16]. We have chosen tiers because they can be specified algebraically. In this section we show that they are also as expressive, in that the set of $P$-serial histories for an arbitrary partition $P$ is equal to the set of $P_i$-serial histories for some tiered partition $P_i$. As such, a required data consistency condition that is defined in terms of $P$-serializability for some partition $P$ can, with a little extra effort, be defined in terms of $P_i$-serializability for some tiered partition $P_i$.

#### A. $P$-definability

Let $T = \{T_1, \ldots, T_n\}$ be a transaction system and let $P = \{P_1, \ldots, P_n\}$ be any partition (not necessarily a tiered partition) of $T$. Then, a history $h$ of $T$ is $P$-serializable if and only if a history $h'$ of $T$ is $P$-serializable, where $h'$ is $P$-serializable if and only if $h$ is $P$-serializable. A history $h$ of $T$ is $P$-serializable if and only if it consists of precisely the $P$-serial histories.

#### B. Slices

Let $h$ be a history. Then, by choosing steps at which changes of transaction occur in the history, we can find steps $s_1 \leq h\ s_2 \leq h \cdots \leq h\ s_{2i-1} \leq h\ s_{2i} \leq h \cdots \leq h\ s_{2m-1} \leq h\ s_{2m}$ such that:

(i) for $1 \leq i \leq m$, all steps in $\{ s : s_{2i-1} \leq h\ s_{2i} \}$ belong to the same transaction;

(ii) for $1 \leq i \leq m$, all steps in $\{ s : s_{2i-1} \leq h\ s_{2i} \}$ and $\{ s : s_{2i+1} \leq h\ s \leq h\ s_{2i+2} \}$ belong to different transactions;

The subsequence of $h$ comprising the steps in $\{ s : s_{2i-1} \leq h\ s \leq h\ s_{2i} \}$ is called the $i$-th slice of $h$ ($1 \leq i \leq m$). Slices are used in the following convenient characterization of $P$-serial histories:

**Lemma 1** Let $T$ be a transaction system and $P$ a partition of $T$. A history $h$ of $T$ is $P$-serializable if and only if each slice of $h$ is a union of complete $P$-segments of the same transaction.

**Proof** If a segment $P_1$ intersects two distinct slices, there will be an intermediate slice with steps belonging to a segment $P_2$ of some other transaction. Thus, there will be $s_{11}, s_{12} \in P_1$ and $s_2 \in P_2$ such that

$$s_{11} \leq h\ s_2 < h\ s_{12}$$

Thus, $h$ is not $P$-serializable by the definition in III-A. This proves the “only if” part of the lemma. The “if” part is obvious. $\blacksquare$

#### C. Tier-definability

A set of histories $H$ of a transaction system $T$ is tier-definable if there is a tiered partition $P_i$ of $T$ such that $H$ is $P_i$-definable. The next theorem shows that general $P$-serial histories can be generated by (a suitable choice of) tiers. This proves the generality of the tiered model.

**Theorem 2** Let $T$ be a transaction system and $P$ a partition of $T$. A set of histories $H$ of $T$ that is $P$-definable is also tier-definable.
Proof Suppose that \( \mathcal{H} \) is a set of histories that are \( P \)-definable. Define the relation \( \sim \), on a transaction \( T_i \in T \) to be such that \( s_1 \sim s_2 \) if, for all \( h \in \mathcal{H} \),

\[
 s_1 \leq_h s \leq_h s_2 \vee s_2 \leq_h s \leq_h s_1 \Leftrightarrow s \in T_i
\]

i.e. \( s_1 \) and \( s_2 \) cannot be separated by a step \( s \) not in \( T_i \) in any history \( h \in \mathcal{H} \). Firstly, we check that \( \sim \) is an equivalence relation on \( T_i \). It is trivially reflexive and symmetric. Suppose that \( s_1 \sim s_2 \) and \( s_2 \sim s_3 \). Let \( h \in \mathcal{H} \) and \( s \) be such that \( s_1 \leq_h s \leq_h s_3 \). If \( s_2 \leq_h s \), then \( s_2 \leq_h s_3 \) and so \( s \in T_i \) as \( s_2 \sim s_3 \). If \( s <_h s_2 \), then \( s_1 \leq_h s <_h s_2 \) and so \( s \in T_i \) as \( s_1 \sim s_2 \). By a symmetric argument \( s_3 \leq_h s \leq_h s_1 \Rightarrow s_1 \in T_i \). It follows that \( \sim \) is transitive and thus an equivalence relation.

Next, we show that the equivalence classes define a tiered partition of \( T \). Let \( T_{iq} \) and \( T_{ir} \) be distinct equivalence classes and assume, on the contrary, that condition (1) of II.B is not satisfied for \( T_{iq} \) and \( T_{ir} \). Then, there are \( s_1, s_2 \in T_{iq} \), \( s_3, s_4 \in T_{ir} \) such that

\[
 s_1 <_i s_3 \text{ and } s_4 <_i s_2
\]

As \( s_1 \) and \( s_3 \) belong to distinct equivalence classes and therefore \( s_1 \not<_i s_3 \), we can choose \( h \in \mathcal{H} \) such that there exists \( s \notin T_i \) with

\[
 s_1 \leq_h s \leq_h s_3
\]

If \( s \leq_h s_2 \) then

\[
 s_1 \leq_h s \leq_h s_2
\]

and so \( s_1 \not<_i s_2 \) contradicting the fact that \( s_1 \) and \( s_2 \) belong to the same equivalence class \( T_{iq} \). On the other hand, if \( s_2 \leq_h s \) then

\[
 s_4 <_h s_2 \leq_h s \leq_h s_3
\]

as \( s_4 <_i s_2 \) implies \( s_4 <_h s_2 \). This means that \( s_4 \not<_i s_3 \) contrary to \( s_3 \) and \( s_4 \) being in the same equivalence class \( T_{ir} \). We have thus shown that the equivalence classes define a tiered partition, \( P_t \), say, of \( T \).

Finally, it remains to show that \( \mathcal{H} \) is \( P_t \)-definable. Now, as \( \mathcal{H} \) is \( P \)-definable, it consists of the \( P \)-serial histories of \( T \). If \( s_1 \) and \( s_2 \) belong to the same \( P \)-segment of \( T \), then, given a history \( h \in \mathcal{H} \) as \( h \) is \( P \)-serial, it is easy to see from the definition of \( P \)-serial histories in III.A that no step \( s \notin T_i \) can come between \( s_1 \) and \( s_2 \) in \( h \). Therefore, by the definition of \( \sim \), \( s_1 \sim s_2 \) and \( s_3, s_3 \) and \( s_3, s_4 \) are in the same tier. Thus, each tier is a union of complete \( P \)-segments. Also, given \( h \in \mathcal{H} \), two steps \( s_1 \) and \( s_2 \) in the same tier, of transaction \( T_i \), say, cannot be in different slices otherwise, by the definition of slices in III.B, they could be separated by some \( s \notin T_i \), which would contradict the fact that, being in the same tier, \( s_1 \sim s_2 \). Therefore, the slices of each \( h \in \mathcal{H} \) are unions of complete tiers. It follows that

\[
 h \in \mathcal{H} \iff h \text{ is } P_t \text{-serial} \iff \text{the slices of } h \text{ are a union of complete } P \text{-segments (by Lemma 1)} \iff \text{the slices of } h \text{ are a union of complete tiers} \iff h \text{ is } P_1 \text{-serial (by Lemma 1)}
\]

IV. ALGEBRAIC THEORY

The basis of the algebraic approach is the observation that sometimes an equivalent history is produced if two adjacent steps are switched. Thus

\[
 r_i^x w_i^y r_j^z w_j^x \text{ and } r_i^x w_i^y r_j^z w_j^z \quad (x \neq y, i \neq j)
\]

are equivalent. For general serializability, not all serializable histories can be derived from serial histories using this form of commutativity. The problem is with commuting write steps. For example, in

\[
 r_i^x w_i^y r_j^y w_j^x \text{ and } r_i^x w_i^y r_j^x w_j^x \quad (x \neq y, i \neq j, j \neq k, i \neq k)
\]

the two writes \( w_i^y \) and \( w_j^x \) may be commuted to form an equivalent history, whereas equivalence will not be preserved if they are commuted in

\[
 r_i^x w_i^y r_j^y w_j^x \quad (x \neq y, i \neq j, j \neq k, i \neq k)
\]

as \( r_i^x \) will, in general, read a different value of \( x \). The solution we shall adopt in such cases is essentially to commute ‘write blocks’ where each write block comprises a write step followed by the read steps that read that write. Thus, the last history is equivalent to

\[
 r_i^x w_i^y r_j^y w_j^x \quad (x \neq y, i \neq j, j \neq k, i \neq k)
\]

where the write blocks \( w_i^y \) and \( w_j^x \) have been commuted.

The theory we present enables all tiered-serializable histories to be derived from tiered-serial histories. In IV.A, we give an equational presentation of the theory in the style of [6]. In IV.B, we show that the algebra of tiered-serializable histories is ‘initial’ for this presentation, which means that histories defined to be tiered-serializable as in II.C are exactly those that can be proved to be tiered-serializable by the equations in IV.A.

A. Equational presentation

In the theory below, the operation \( r \) and \( w \) generate read and write steps \( r(j,1,x) \) and \( w(j,1,x) \) where \( j \), \( 1 \) and \( x \) are natural number labels for tiers, transactions and data items respectively. Natural numbers are specified by the operations \( 1 \), \( \text{succ} \) and \( \text{isequal} \). Histories are formed by use of the associative infix operator . to concatenate other histories. For example, the following expression:

\[
 r(1, \text{succ}(1), \text{succ}(\text{succ}(1))).w(1, \text{succ}(1), \text{ succ}(\text{succ}(1)))
\]

represents the history comprising a read step followed by a write step to a data item labelled 3 by a tier labelled 1 of a transaction labelled 2. The predicate operations \( \text{tier} \), \( \text{trans} \) and \( \text{tsaseq} \) have the following meanings: \( \text{tier}(h,j,i) \) asserts that \( h \) is a \( j \)-labelled tier of an \( i \)-labelled transaction, \( \text{trans}(h,m,i) \) asserts that \( h \) is an \( i \)-labelled transaction with \( m \) tiers, and \( \text{tsaseq}(h,n) \) asserts that \( h \) is a tiered-serial history with \( n \) transactions.

Equations (E1)-(E4) belong to the specification of natural numbers, and (E5) is associativity of the . concatenation operator. Equations (E6) – (E13) enable all serial histories of transactions to be generated, where the constituent transactions
are labelled by consecutive integers starting at 1, and where the tiers within each transaction are also labelled consecu-

tively. The equation (E14) allows other tiered-serial histories, where transactions and tiers are not necessarily labelled by consecutive integers, to be generated by commuting adjacent tiers belonging to different transactions (tiers h3 and h4 are commuted in (E14)).

Proving serializability for a history h requires proving tserz(h) = TRUE. The predicate tserz and history equivalence predicates Tequiv and Vequiv along with corresponding equations (E15)-(E34) are discussed in IV.B.

SORTS

N (naturals), H (histories)

VARIABLES

m, n, i, j, l1, l2, ..., j1, j2, ... , x, y : N
h, h1, h2, ... : H

OPERATIONS

(O1) 1 : -> N
(O2) succ : N -> N
(O3) r : N, N, N -> H
(O4) w : N, N, N -> H
(O5) . : H, H -> H
(O6) _is_ : N, N -> B
(O7) _not_ : H, H -> B
(O8) tier : H, N, N -> B
(O9) trans : H, N, N -> B
(O10) tser : H, N -> B
(O11) wb : H, N -> B
(O12) Tequiv : H, H -> B
(O13) Vequiv : H, H -> B
(O14) tserz : H -> B

EQUATIONS

(E1) 1 isequal 1 = TRUE
(E2) 1 isequal succ(n) = FALSE
(E3) m isequal n = n isequal m
(E4) m isequal n = succ(m) isequal succ(n)
(E5) (h1.h2).h3 = h1.(h2.h3)
(E6) tier(r(j,i,x), j, i) = TRUE
(E7) tier(w(j,i,x), j, i) = TRUE
(E8) tier(h.r(j,i,x), j, i) = TRUE
(E9) tier(h.w(j,i,x), j, i) = TRUE
(E10) trans(h, l, i) = TRUE
(E11) trans(h1.h2, succ(m), i) = TRUE
(E12) tser(h, l, i) = TRUE
(E13) tser(h1.h2, succ(n)) = TRUE
(E14) tser(h1.h2.h4.h3.h5.h6.n) = TRUE

B. Initiality

The main result is Theorem 7 which shows that a history h is tiered-serializable if and only if tserz(h) = TRUE can be proved for the term h corresponding to h. Proving tserz(h) = TRUE requires proving Tequiv(h1,h2) = TRUE and Vequiv(h1,h2) = TRUE by (E34) for some history h1. To this end, we first define the two new equivalences for

\[
\text{tserz}(h) = \begin{cases} 
\text{true} & \text{if tser(h1,n) = true, Tequiv(h1,h2) = true, and Vequiv(h1,h2) = true} 
\end{cases}
\]
histories, namely ‘T-equivalence’ and ‘V-equivalence’ which, together amount to history equivalence as defined in ILC, and show that they correspond to \( \text{Tequiv} \) and \( \text{Vequiv} \) in the equational presentation in IVA respectively. The proof that T-equivalence corresponds to \( \text{Tequiv} \) is immediate. The proof that V-equivalence corresponds to \( \text{Vequiv} \) is more difficult and is proved in Lemma 6, after defining a relation \( \approx^V \) on histories and proving intermediate results (Lemmas 3, 4 and 5) concerning \( \approx^V \) and V-equivalence.

Two histories \( h_1 \) and \( h_2 \) are \( T \)-equivalent if they are histories of the same transaction system. It is easy to see that \( h_1 \) and \( h_2 \) are \( T \)-equivalent if and only if one can be obtained from the other by repeatedly commuting adjacent steps belonging to different transactions, i.e.

\[
h_1, h_2 \text{ are } T \text{-equivalent } \iff \text{Tequiv}(h_1, h_2) = \text{TRUE} \quad (3)
\]

where \( \text{Tequiv}(h_1, h_2) = \text{TRUE} \) can be proved for the terms \( h_1 \) and \( h_2 \) corresponding to \( h_1 \) and \( h_2 \) respectively, from the equations (E25) – (E27).

Two histories \( h_1 \) and \( h_2 \) comprising the same set of steps but not necessarily having steps of a given transaction in the same order, are \( V \)-equivalent if they have the same \( \text{sees} \) function and the same last write steps to each variable \( x \). It is easy to see that \( h_1 \) and \( h_2 \) are \( V \)-equivalent (in the sense of ILC) if and only if \( h_1 \) and \( h_2 \) are \( T \)-equivalent and \( V \)-equivalent.

We characterize V-equivalence in terms of the \( \approx^V \) relation given below which requires the definition of a ‘write block’.

A write block to a variable \( x \) is a sequence of steps comprising a write step to \( x \), followed by several reads to \( x \). A history \( h \) is in write block form iff it is of the form

\[
w^{x_1} \ldots w^{x_{p_1}} \ldots w^{x_{p_2}} \ldots w^{x_2} \ldots w^{x_1} \ldots (p \geq 1)
\]

where the square brackets indicate that there may or may not be read steps after the last write step \( w^{x_1} \). The relation \( \approx^V \) is the transitive closure of the reflexive and symmetric relation \( \approx^V \) on histories which has \( h_1 \approx^V h_2 \) iff any of the following is the case:

\[
h_1 = s_1 \ldots s_k = r_1^{y_1} r_2^{y_2} s_{k+2} \ldots s_K, \quad (4)
\]

\[
h_2 = s_1 \ldots s_k = r_1^{y_1} r_2^{y_2} s_{k+2} \ldots s_K, \quad (x \neq y) \quad (5)
\]

\[
h_1 = s_1 \ldots s_k = w_1^{x_1} \ldots w_2^{x_2} \ldots w_{k+2} \ldots s_K, \quad (x \neq y) \quad (6)
\]

\[
h_1 = s_1 \ldots s_k = w_1^{x_1} \ldots w_2^{x_2} \ldots w_{k+2} \ldots s_K, \quad (x \neq y) \quad (7)
\]

Thus, for histories \( h_1 \) and \( h_2, h_1 \approx^V h_2\) iff one can be obtained from the other by commuting adjacent non-conflicting steps ((4), (5) and (6)), or adjacent write blocks to a variable \( x \), which immediately precede a write \( w_1^{x_1} \) to \( x \) as in (7). As (4), (5), (6) and (7) correspond to (E30), (E31), (E32) and (E33) respectively

\[
h_1 \approx^V h_2 \iff \text{Vequiv}(h_1, h_2) = \text{TRUE} \quad (8)
\]

for the terms \( h_1 \) and \( h_2 \).

Lemma 3 If \( h \approx^V h' \), then \( h \) and \( h' \) have the same last write step to any given variable \( x \).

Proof The last write step to \( x \) in \( h \) cannot be moved to the left of another write step to \( x \) by use of any of (4), (5), (6) or (7).

Lemma 4 Let \( h \) be a history of the form

\[
h = s_1 \ldots s_k w^{x_1} \ldots w^{x_{p_1}} \ldots w^{x_{p_2}} \ldots w^{x_2} s_{k-1} s_k
\]

where \( s_{k-1} \) is a write step to a variable \( x \) and \( n \) write blocks to \( x \) precede \( s_{k-1} \) in \( h \). Then, given a permutation \( \pi \) of \( \{1, \ldots, n\} \), for

\[
h' = s_1 \ldots s_k w^{x_1} \ldots w^{x_{p_1}} \ldots w^{x_{p_2}} \ldots w^{x_2} s_{k-1} s_k
\]

we have that \( h \approx^V h' \).

Proof The history \( h' \) can be obtained by repeatedly commuting adjacent write blocks, in the manner of (7), until the desired order of write blocks is achieved.

Lemma 5 Let \( h_1 \) and \( h_2 \) be histories. Then, \( h_1 \) and \( h_2 \) are \( V \)-equivalent if and only if \( h_1 \approx^V h_2 \).

Proof If \( h_1 \approx^V h_2 \), then \( h_1 \) and \( h_2 \) have the same \( \text{sees} \) function as (4)-(6) do not affect the write that a read step sees. Also, by Lemma 3, \( h_1 \) and \( h_2 \) have the same last write step to any given variable \( x \). Thus, \( h_1 \) and \( h_2 \) are \( V \)-equivalent. This proves the “if” part.

We now prove the “only if” part. Suppose that \( h_1 \) and \( h_2 \) are \( V \)-equivalent. By repeatedly commuting adjacent \( w^{x_1} \) and \( w^{y_1} \) steps in \( h_1 \) and \( h_2 \), as in (5), we can find histories \( h_1' \) and \( h_2' \) in write block form such that

\[
h_1 \approx^V h_1', \quad h_2 \approx^V h_2'
\]

say

\[
h_1' = w^{x_1} r_1^{y_1} \ldots w^{x_{p_1}} r_1^{y_{p_1}} \ldots w^{x_{p_2}} r_1^{y_{p_2}} \ldots w^{x_{k-2}} s_{k-1} s_k
\]

\[
h_2' = w^{y_1} r_1^{x_1} \ldots w^{y_{p_1}} r_2^{x_{p_1}} \ldots w^{y_{p_2}} r_2^{x_{p_2}} \ldots w^{y_{k-2}} s_{k-1} s_k
\]

Since \( h_1 \approx^V h_1' \), by the “if” part of this lemma \( h_1 \) is \( V \)-equivalent to \( h_1' \). Likewise, \( h_2 \) is \( V \)-equivalent to \( h_2' \). As \( h_1 \) is \( V \)-equivalent to \( h_2 \) and \( V \)-equivalence is clearly transitive, it follows that \( h_1' \) is \( V \)-equivalent to \( h_2' \). From this it is clear that \( \rho = q \) and, indeed, \( h_1' \) and \( h_2' \) have the same write blocks though possibly in a different order. Furthermore, \( h_1' \) and \( h_2' \) have the same last write steps to any given variable \( x \).

Applying (4), (5) and (6) repeatedly we can find \( h_1'' \) and \( h_2'' \) such that \( h_1'' \approx^V h_1', h_2'' \approx^V h_2' \) and all write blocks to a given variable \( x \) are grouped together in \( h_1'' \) and \( h_2'' \). Let the two sequences of (distinct) variables corresponding to these groups of write blocks in \( h_1'' \) and \( h_2'' \) be:

\[
x_1^{(1)} \ldots x_{p_1}^{(1)} \quad \text{and} \quad x_1^{(2)} \ldots x_{p_2}^{(2)}
\]
respectively. By further applying (4), (5) and (6) a multitude of times, effectively to commute adjacent whole groups of write blocks to different variables, we can find \( h''_1 \) and \( h''_2 \) such that \( h''_1 \equiv V h''_1 \) and \( h''_2 \equiv V h''_2 \) and the two sequences of (distinct) variables corresponding to the groups of write blocks in \( h''_1 \) and \( h''_2 \): 

\[
y^{(1)}_1 \ldots y^{(1)}_{p_1} \text{ and } y^{(2)}_1 \ldots y^{(2)}_{p_2}
\]

are the same for \( h''_1 \) and \( h''_2 \), i.e. \( y^{(1)}_k = y^{(2)}_k \) \((1 \leq k \leq p_2)\).

The only difference between \( h''_1 \) and \( h''_2 \) is the order in which write blocks occur within a write block grouping corresponding to a variable \( x \). As noted above, \( h_1 \) and \( h_2 \) have the same last write blocks to \( x \). It is clear from this and the construction of \( h''_1 \), \( h''_2 \), \( h''_1 \) and \( h''_2 \), that \( h''_1 \) and \( h''_2 \) have the same last write blocks to any given \( x \). The other write blocks to this \( x \) in \( h''_1 \) and \( h''_2 \) are just a permutation, \( \pi_x \), say, of the remaining write blocks to \( x \) in \( h''_2 \). By repeated use of Lemma 4 for each variable \( x \) in turn we deduce that \( h''_1 \equiv V h''_2 \).

To summarize,

\[
h_1 \approx V h_1 \approx V h''_1 \approx V h''_2 \approx V h_2 \approx V h_2
\]

It follows that \( h_1 \approx V h_2 \).

**Lemma 6** Let \( h_1 \) and \( h_2 \) be histories. Then, \( h_1 \) and \( h_2 \) are \( V \)-equivalent if and only if \( V \equiv \text{tser}(h_1, h_2) \) is true can be proved for the terms \( h_1 \) and \( h_2 \) corresponding to \( h_1 \) and \( h_2 \) respectively.

**Proof** Follows from (8) and Lemma 5.

**Theorem 7** A history is tiered-serializable if and only if \( tser(h) = \text{true} \) can be proved for the term \( h \) corresponding to \( h \).

**Proof**

\[ h \text{ tiered-serializable} \iff h \text{ equivalent to tiered-serial history } h \]

\[ h \text{ tiered-serial, } h \text{ and } h \text{ } \]

\[ h \text{ and } h \text{ } \]

\[ h \approx V h \]

(by Lemma 5)

\[ \text{tser}(h_1, n) = \text{TRUE for some } n, \]

\[ \text{tser}(h_1, n) = \text{TRUE, } \]

\[ \text{Vequiv}(h_1, h) = \text{TRUE, } \]

\[ \text{tser}(h_1, n) = \text{TRUE for some } n, \]

\[ \text{tser}(h_1, n) = \text{TRUE, } \]

\[ \text{tser}(h_1, n) = \text{TRUE for some } n, \]

(by (3) and Lemma 6)

\[ \text{tser}(h) = \text{TRUE (by (E34))} \]

**C. Example**

We give an example of an algebraic proof of the tiered-serializability of a history \( h_1 \) which cannot be achieved by commuting non-conflicting steps. The proof derives a tiered-serial history \( h \) from \( h_1 \) using equations (E1) – (E34) of IV.A.

For \( i = 1, 2, 3 \), let the transaction \( T_i \) be given by:

and the history \( h_1 \) of \( \{T_1, T_2, T_3\} \) be \( h_1 = \)

\[ r_1^{11} r_2^{12} r_3^{13} w_1^{11} w_2^{12} w_3^{13} r_1^{21} r_2^{22} r_3^{23} w_1^{21} w_2^{22} w_3^{23} \]

The notation for steps has been condensed, e.g. \( r(j, i, x) \) becomes \( r_{ji}^x \), and blocks of steps of interest are underlined. Now, by repeatedly commuting write steps to different data items ((E32), (E23) and (E24)), we have that

\[ \text{Vequiv}(w_1^{11}, w_1^{12}, w_2^{12}, w_1^{21}, w_2^{21}, w_3^{23}) = \text{TRUE} \]

(9)

By commuting the write blocks comprising the single steps \( w_1^{11} \) and \( w_2^{12} \) ((E33), (E23) and (E24)) in the second term in (9), we have that

\[ \text{Vequiv}(w_1^{11}, w_1^{12}, w_2^{12}, w_1^{21}, w_2^{21}, w_3^{23}) = \text{TRUE} \]

(10)

By commuting write steps ((E32), (E23) and (E24)),

\[ \text{Vequiv}(w_1^{11}, w_1^{12}, w_2^{12}, w_1^{21}, w_2^{21}, w_3^{23}) = \text{TRUE} \]

(11)

By (9), (10) and (11) and transitivity of \( \text{Vequiv} (\text{(E22)}), \)

\[ \text{Vequiv}(w_1^{11}, w_1^{12}, w_2^{12}, w_1^{21}, w_2^{21}, w_3^{23}) = \text{TRUE} \]

(12)

Thus, by (12), (E23) and (E24),

\[ \text{Vequiv}(h_2, h_1) = \text{TRUE} \]

(13)

where \( h_2 \) is the history \( h_2 = \)

\[ r_1^{11} r_2^{12} r_3^{13} w_1^{11} w_2^{12} w_3^{13} r_1^{21} r_2^{22} r_3^{23} w_1^{21} w_2^{22} w_3^{23} \]

By repeatedly commuting non-conflicting read and write steps ((E31), (E30), (E23) and (E24)),

\[ \text{Vequiv}(h_2, h_1) = \text{TRUE} \]

(14)

where \( h \) is the history \( h = \)

\[ r_1^{11} r_2^{12} r_3^{13} w_1^{11} w_2^{12} w_3^{13} r_1^{21} r_2^{22} r_3^{23} w_1^{21} w_2^{22} w_3^{23} \]

By (13), (14) and transitivity of \( \text{Vequiv} (\text{(E22)}), \)

\[ \text{Vequiv}(h_2, h_1) = \text{TRUE} \]

(15)

By straightforward applications of (E5)–(E14), \( h \) is clearly tiered-serial, i.e.

\[ \text{tser}(h, 3) = \text{TRUE} \]

(16)

By repeatedly commuting steps belonging to different transactions we have, by (E25)–(E27) and (E15)–(E19), that

\[ \text{Vequiv}(h_2, h_1) = \text{TRUE} \]

(17)

By (16), (17), (15) and (E34), we conclude that \( \text{tser}(h_1) = \text{TRUE} \) as required.

**V. Conclusions**

We have given an algebraic theory of serializable schedules for a model of concurrent partitioned transactions. The main contribution of this work lies in its generality. Rather than presenting a neat algebraic method for a specific class of serializability problem, we have given a general algebraic method that is applicable to a very wide range of serializability problems. Practically, an algebraic approach may or may not be the best way of proving serializability in a given application. It is not the aim of this paper to determine if
an algebraic approach is suitable for a particular application. The aim in this paper is to give the option of an algebraic approach if an alternative method of proof is required.

Most other work on algebraic specification of concurrency has been in the area of process algebras, notably CSP [7] and the $\pi$-calculus [15]. A comprehensive survey of different algebraic approaches to concurrency can be found in [1], although most approaches are applied to processes other than database transactions. An exception is the equational theory in [3] which is used to analyze the interplay between ACID properties of transactions. The initial-algebra approach in this paper has not been used in connection with database transactions. However, it is an established area of algebraic specification and standardized systems to support its use have been developed recently [2].

**REFERENCES**


