Algebraic Specification of Serializability for Partioned 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 beyond commutativity-based serializability. It should be noted that any specification of serializability in propositional temporal logic has a proof method by virtue of an axiomatization of the logic and, as it is decidable, a fully automatic algorithm. However, such an algorithm cannot improve on one based on dependency graphs as such logics are PSPACE-complete [18].

In this paper, we give an algebraic method for proving general serializability. There are two main results. Firstly, an extended partitioned transaction model is given and is shown to be as expressive as the general partitioned transaction model. Secondly, an algebraic method is given for proving general serializability in our extended model. Although general serializability is not commutative, we are able to give an initial-algebra specification of serializable schedules in the style of [6] making use of conditional equations to overcome the problem. The paper is structured as follows. Section II gives our extended model of partitioned transactions. In Section III it is proved to be as expressive as the general partitioned transaction model. The algebraic specification of serializable schedules for our extended transaction model is in Section IV. The conclusions are in Section V.

II. A MODEL OF PARTITIONED TRANSACTIONS

A. Steps, transactions and histories

A read or write step to $x$ in transaction $T_i$ will be denoted $r(i,x)$ or $w(i,x)$ respectively. A more condensed notation will be used in places in Section IV where $r_i^{\tau}$ and $w_i^{\tau}$ will replace $r(i,x)$ and $w(i,x)$. The subscript or superscript will also be omitted occasionally when of no interest. A transaction $(T_i; \preceq_i)$ is a finite partially ordered set of steps with transaction identifier $i$. The corresponding irreflexive order is denoted $<_i$. A transaction system is a finite set of transactions.

A schedule or history $h$ is a sequence of steps. The total order of the sequence is denoted $\leq_h$ with $<_h$ as the irreflexive

\begin{itemize}
  \item \textbf{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.

\textbf{Keywords}—Algebraic Specification, Partitioned Transactions, Serializability.

\section{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 beyond commutativity-based serializability. It should be noted that any specification of serializability in propositional temporal logic has a proof method by virtue of an axiomatization of the logic and, as it is decidable, a fully automatic algorithm. However, such an algorithm cannot improve on one based on dependency graphs as such logics are PSPACE-complete [18].

In this paper, we give an algebraic method for proving general serializability. There are two main results. Firstly, an extended partitioned transaction model is given and is shown to be as expressive as the general partitioned transaction model. Secondly, an algebraic method is given for proving general serializability in our extended model. Although general serializability is not commutative, we are able to give an initial-algebra specification of serializable schedules in the style of [6] making use of conditional equations to overcome the problem. The paper is structured as follows. Section II gives our extended model of partitioned transactions. In Section III it is proved to be as expressive as the general partitioned transaction model. The algebraic specification of serializable schedules for our extended transaction model is in Section IV. The conclusions are in Section V.

\section{II. A MODEL OF PARTITIONED TRANSACTIONS}

\textbf{A. Steps, transactions and histories}

A read or write step to $x$ in transaction $T_i$ will be denoted $r(i,x)$ or $w(i,x)$ respectively. A more condensed notation will be used in places in Section IV where $r_i^{\tau}$ and $w_i^{\tau}$ will replace $r(i,x)$ and $w(i,x)$. The subscript or superscript will also be omitted occasionally when of no interest. A transaction $(T_i; \preceq_i)$ is a finite partially ordered set of steps with transaction identifier $i$. The corresponding irreflexive order is denoted $<_i$. A transaction system is a finite set of transactions.

A schedule or history $h$ is a sequence of steps. The total order of the sequence is denoted $\leq_h$ with $<_h$ as the irreflexive

\begin{itemize}
  \item \textbf{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.

\textbf{Keywords}—Algebraic Specification, Partitioned Transactions, Serializability.

\section{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 beyond commutativity-based serializability. It should be noted that any specification of serializability in propositional temporal logic has a proof method by virtue of an axiomatization of the logic and, as it is decidable, a fully automatic algorithm. However, such an algorithm cannot improve on one based on dependency graphs as such logics are PSPACE-complete [18].

In this paper, we give an algebraic method for proving general serializability. There are two main results. Firstly, an extended partitioned transaction model is given and is shown to be as expressive as the general partitioned transaction model. Secondly, an algebraic method is given for proving general serializability in our extended model. Although general serializability is not commutative, we are able to give an initial-algebra specification of serializable schedules in the style of [6] making use of conditional equations to overcome the problem. The paper is structured as follows. Section II gives our extended model of partitioned transactions. In Section III it is proved to be as expressive as the general partitioned transaction model. The algebraic specification of serializable schedules for our extended transaction model is in Section IV. The conclusions are in Section V.

\section{II. A MODEL OF PARTITIONED TRANSACTIONS}

\textbf{A. Steps, transactions and histories}

A read or write step to $x$ in transaction $T_i$ will be denoted $r(i,x)$ or $w(i,x)$ respectively. A more condensed notation will be used in places in Section IV where $r_i^{\tau}$ and $w_i^{\tau}$ will replace $r(i,x)$ and $w(i,x)$. The subscript or superscript will also be omitted occasionally when of no interest. A transaction $(T_i; \preceq_i)$ is a finite partially ordered set of steps with transaction identifier $i$. The corresponding irreflexive order is denoted $<_i$. A transaction system is a finite set of transactions.

A schedule or history $h$ is a sequence of steps. The total order of the sequence is denoted $\leq_h$ with $<_h$ as the irreflexive
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 $\mathcal{P}_i = \{T_{i1}, \ldots, T_{ip}\}$ is a partition of the set of steps of $T_i$ ($1 \leq i \leq n$). 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

$$s_1, s_2 \in T_{iq} \quad s_3, s_4 \in T_{ir}$$

and

$$s_1 <_i s_3 \quad \text{and} \quad s_4 <_i s_2 \quad (1)$$

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 II.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 $s_k$ of a variable $x$ sees a write step $s_l$ to the same variable $x$ in $h$, denoted functionally as $sees_h(s_k) = s_l$, iff

$$s_l <_h s_k$$

and there is no write step $s$ to $x$ such that

$$s_l <_h s <_h s_k$$

A history $h'$ is equivalent to $h$, denoted $h' \approx h$, iff there is a permutation $\rho$ of $\{1, \ldots, K\}$ such that $h'$ is

$$s_{\rho(1)} \ldots s_{\rho(K)}$$

and

$$sees_h(s_k) = s_l \Leftrightarrow sees_{h'}(s_k) = s_l$$

for every read step $s_k$ and write step $s_l$ and, furthermore, for each variable $x$, $h$ and $h'$ 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

$$s_l <_h s_2 \quad \text{for all } s_1 \in T_{i_1}, s_2 \in T_{i_2} \quad \text{or} \quad (2)$$

$$s_2 <_h s_1 \quad \text{for all } s_1 \in T_{i_1}, s_2 \in T_{i_2}$$

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 $T_{i_1,j_1}, T_{i_2,j_2}$, either

$$s_l <_h s_2 \quad \text{for all } s_1 \in T_{i_1,j_1}, s_2 \in T_{i_2,j_2} \quad \text{or} \quad (3)$$

$$s_2 <_h s_1 \quad \text{for all } s_1 \in T_{i_1,j_1}, s_2 \in T_{i_2,j_2}$$

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 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:

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

Consider the following history for the concurrent execution of three such transactions $T_1, T_2$ and $T_3$:

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

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

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
A. \(\mathcal{P}\)-definability

Let \(T = \{T_1, \ldots, T_n\}\) be a transaction system and let \(\mathcal{P} = \{P_1, \ldots, P_m\}\) be any partition (not necessarily a tiered partition) of \(T\). Then, a history \(h\) of \(T\) is \(\mathcal{P}\)-serial (as in [16]) iff given distinct transactions \(T_{i_1}\) and \(T_{i_2}\) and segments \(P_1\) in \(\mathcal{P}_{i_1}\) and \(P_2\) in \(\mathcal{P}_{i_2}\) either

\[
s_1 <_h s_2 \quad \text{for all } s_1 \in P_1, \ s_2 \in P_2
\]

or

\[
s_2 <_h s_1 \quad \text{for all } s_1 \in P_1, \ s_2 \in P_2
\]

A history \(h\) of \(T\) is \(\mathcal{P}\)-serializable iff it is equivalent to a \(\mathcal{P}\)-serial history. A set of histories \(\mathcal{H}\) of \(T\) is \(\mathcal{P}\)-definable iff it consists of precisely the \(\mathcal{P}\)-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

\[
s_1 <_h s_2 <_h \cdots <_h s_{2i-1} <_h s_{2i} <_h \cdots <_h s_{2m-1} <_h s_{2m}
\]

such that:

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

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

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

Lemma 1 Let \(T\) be a transaction system and \(\mathcal{P}\) a partition of \(T\). A history \(h\) of \(T\) is \(\mathcal{P}\)-serial if and only if each slice of \(h\) is a union of complete \(\mathcal{P}\)-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

\[
s_{11} <_h s_2 <_h s_{12}
\]

Thus, \(h\) is not \(\mathcal{P}\)-serial by the definition in III.A. This proves the “only if” part of the lemma. The “if” part is obvious.

C. Tier-definability

A set of histories \(\mathcal{H}\) of a transaction system \(T\) is tier-definable iff there is a tiered partition \(\mathcal{P}_t\) of \(T\) such that \(\mathcal{H}\) is \(\mathcal{P}_t\)-definable. The next theorem shows that general \(\mathcal{P}\)-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 \(\mathcal{P}\) a partition of \(T\). A set of histories \(\mathcal{H}\) of \(T\) that is \(\mathcal{P}\)-definable is also tier-definable.
Proof Suppose that $\mathcal{H}$ is a set of histories that is $\mathcal{P}$-definable. Define the relation $\sim$, on a transaction $T_i \in T$ to be such that $s_1 \sim s_2$ iff, for all $h \in \mathcal{H}$,
\[
s_1 \leq_h s \leq h s_2 \lor s_2 \leq h s \leq s_1 \Rightarrow s \in T_i
\]
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 $s_1 \leq h s \leq s_3$. If $s_2 \leq h s$, then $s_2 \leq h s \leq s_3$ and so $s \in T_i$ as $s_2 \sim s_3$. If $s < h s$, then $s_1 \leq h s \leq s_2$ and so $s \in T_i$ as $s_1 \sim s_2$. By a symmetric argument $s_3 \leq h s \leq s_1 \Rightarrow s_1 \in T_i$. 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
\[
s_1 < s_3 \text{ and } s_4 < s_2
\]
as $s_1$ and $s_3$ belong to distinct equivalence classes and therefore $s_1 \not\sim s_3$, 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 $s_1 \not\sim s_2$ contradicting the fact that $s_1$ and $s_2$ belong to the same equivalence class $T_{iq}$. On the other hand, if $s_2 < h s$ then
\[
s_4 < h s_2 < h s \leq h s_3
\]
as $s_4 < s_2$ implies $s_4 < h s_2$. This means that $s_4 \not\sim s_3$ 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, $\mathcal{P}_l$, say, of $T$.

Finally, it remains to show that $\mathcal{H}$ is $\mathcal{P}_l$-definable. Now, as $\mathcal{H}$ is $\mathcal{P}$-definable, it consists of the $\mathcal{P}$-serial histories of $T$. If $s_1$ and $s_3$ belong to the same $\mathcal{P}$-segment of $T$, then, given a history $h \in \mathcal{H}$, $h$ is $\mathcal{P}$-serial and is easy to see from the definition of $\mathcal{P}$-serial histories in III.A that no step $s \notin T_i$ can come between $s_1$ and $s_3$ in $h$. Therefore, by the definition of $\sim$, $s_1 \sim s_3$ and $s_1$ and $s_2$ are in the same tier. Thus, each tier is a union of complete $\mathcal{P}$-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
\[
h \in \mathcal{H} \Leftrightarrow h \text{ is } \mathcal{P}_l\text{-serial}
\]
\[
\Leftrightarrow \text{the slices of } h \text{ are a union of complete } \mathcal{P}\text{-segments (by Lemma 1)}
\]
\[
\Leftrightarrow \text{the slices of } h \text{ are a union of complete tiers}
\]
\[
\Leftrightarrow h \text{ is } \mathcal{P}_l\text{-serial (by Lemma 1)}
\]

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
\[
r_1^w r_2^w w_1^z w_2^z \text{ and } r_1^w w_2^w r_2^y r_2^z (x \neq y, i \neq j)
\]
is 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
\[
r_1^w w_1^z w_2^z r_1^z r_k^z (x \neq y, i \neq j, j \neq k, i \neq k)
\]
the two writes $w_i^z$ and $w_j^z$ may be commuted to form an equivalent history, whereas equivalence will not be preserved if they are commuted in
\[
r_1^z r_k^z w_1^z w_2^z (x \neq y, i \neq j, j \neq k, i \neq k)
\]
as $r_k^z$ 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
\[
r_1^z r_k^z w_1^z r_k^z w_2^z
\]
where the write blocks $w_1^z$ and $w_2^z$ 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 $\tau$ and $\omega$ generate read and write steps $\tau(j, i, x)$ and $\omega(j, i, x)$ where $j$, $i$ and $x$ are natural number labels for tiers, transactions and data items respectively. Natural numbers are specified by the operations $\text{succ}$ and $\text{isequal}$. Histories are formed by use of the associative infix operator $\cdot$ to concatenate other histories. For example, the following expression:

$$\tau(1, \text{succ}(1), \text{succ}(\text{succ}(1))), \omega(1, \text{succ}(1), \text{succ}(\text{succ}(1)))$$

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 $\text{tser}$ 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 $\text{tser}(h, n)$ 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 $\cdot$ 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 consecutively. 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 $h_3$ and $h_4$ are commuted in (E14)).

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

```plaintext
SORTS
N (naturals), H (histories)

VARIABLES
m,n,i,j,j1,j2,...,j1,j2,... : N
h,h1,h2,... : H

OPERATIONS
(O1) 1 : $\rightarrow$ N
(O2) succ : N $\rightarrow$ N
(O3) r : N,N,N $\rightarrow$ H
(O4) w : N,N,N $\rightarrow$ H
(O5) _ : H,H $\rightarrow$ H
(O6) _isequal_ : N,N $\rightarrow$ B
(O7) _=_ : H,H $\rightarrow$ B
(O8) tier : H,N,N $\rightarrow$ B
(O9) trans : H,N,N $\rightarrow$ B
(O10) tser : H,N $\rightarrow$ B
(O11) wb : H,N $\rightarrow$ B
(O12) Tequiv : H,H $\rightarrow$ B
(O13) Vequiv : H,H $\rightarrow$ B
(O14) tserz : H $\rightarrow$ B

EQUATIONS
(E1) 1 isequal 1 = TRUE
(E2) 1 isequal succ(n) = FALSE
(E3) m isequal n = m isequal succ(m)
(E4) (m isequal n is 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,1,1) = TRUE
(E11) trans(h1.h2,success(m),i) = TRUE
(E12) tser(h,1) = TRUE
(E13) tser(h1.h2,success(m)) = 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 $\text{tser}(h) = \text{TRUE}$ can be proved for the term $h$ corresponding to $h$. Proving $\text{tser}(h) = \text{TRUE}$ requires proving $\text{Tequiv}(h1,h2) = \text{TRUE}$ and $\text{Vequiv}(h1,h2) = \text{TRUE}$ by (E34) for some history $h1$. To this end, we first define the two new equivalences for

(E15) Tequiv(h,h) = TRUE
(E16) Tequiv(h1,h2) = Tequiv(h2,h1)
(E17) Tequiv(h1,h3) = TRUE
IF Tequiv(h1,h2) = TRUE,
Tequiv(h2,h3) = TRUE
(E18) Tequiv(h1,h1,h2) = TRUE
IF Tequiv(h1,h2) = TRUE
TEEquiv(h1,h2) = TRUE
(E19) Tequiv(h1.h2,h2) = TRUE
IF Tequiv(h1,h2) = TRUE

(E20) Vequiv(h,h) = TRUE
(E21) Vequiv(h1,h2) = Vequiv(h2,h1)
(E22) Vequiv(h1,h3) = TRUE
IF Vequiv(h1,h2) = TRUE,
Vequiv(h2,h3) = TRUE
(E23) Vequiv(h1.h1,h2.h2) = TRUE
IF Vequiv(h1,h2) = TRUE
(E24) Vequiv(h1.h2,h2.h2) = TRUE
IF Vequiv(h1,h2) = TRUE

(E25) Tequiv(r(j1,i1,x).r(j2,i2,y),
r(j2,i2,y).r(j1,i1,x)) = TRUE
IF i1 isequal i2 = FALSE
(E26) Tequiv(r(j1,i1,x).w(j2,i2,y),
 w(j2,i2,y).r(j1,i1,x)) = TRUE
IF i1 isequal i2 = FALSE
(E27) Tequiv(w(j1,i1,x).w(j2,i2,y),
 w(j2,i2,y).w(j1,i1,x)) = TRUE
IF i1 isequal i2 = FALSE

(E28) wb(w(j,i,x),x) = TRUE
(E29) wb(h.r(j,i,x),x) = TRUE
IF wb(h,x) = TRUE
(E30) Tequiv(r(j1,i1,x).r(j2,i2,y),
 r(j2,i2,y).r(j1,i1,x)) = TRUE
(E31) Tequiv(r(j1,i1,x).w(j2,i2,y),
 w(j2,i2,y).r(j1,i1,x)) = TRUE
IF x isequal y = FALSE
(E32) Tequiv(w(j1,i1,x).w(j2,i2,y),
 w(j2,i2,y).w(j1,i1,x)) = TRUE
IF x isequal y = FALSE
(E33) Vequiv(h1.h2.w(j,i,x),
 h2.h1.w(j,i,x)) = TRUE
IF wb(h1,x) = TRUE, wb(h2,x) = TRUE
(E34) tserz(h2) = TRUE
IF tser(h1,n) = TRUE,
Tequiv(h1,h2) = TRUE,
Vequiv(h1,h2) = TRUE
histories, namely ‘T-equivalence’ and ‘V-equivalence’, which together amount to history equivalence as defined in ILC, and show that they correspond to Tequiv and Vequiv in the equational presentation in IV.A respectively. The proof that T-equivalence corresponds to Tequiv is immediate. The proof that V-equivalence corresponds to 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 and only if one can be obtained from the other by repeatedly commuting adjacent steps belonging to different transactions, i.e.

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

where Tequiv(h1, h2)=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 a write step to a variable \( x \) preceding \( s_{t-1} \) in \( h \). Then, given a permutation \( \pi \) of \( \{1, \ldots, n\} \), for

\[
h' = s_1 \ldots s_k w^x_{p_1} \ldots w^x_{p_t} \ldots w^x_{p_s} s_{t-1} s_t \ldots s_K \]

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.

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 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 \( w^x_i \) and \( w^y_i \) 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

\[
h_1 \approx V h'_1 \text{, } h_2 \approx V h'_2
\]

say

\[
h'_1 = w^{x_1} r^{x_1}_{p_1} \ldots w^{x_i} r^{x_i}_{p_i} \ldots w^{x_p} r^{x_p}_{p_p} s_{t-1} \ldots s_K;
\]

\[
h'_2 = w^{y_1} r^{y_1}_{p_1} \ldots w^{y_i} r^{y_i}_{p_i} \ldots w^{y_p} r^{y_p}_{p_p} s_{t-1} \ldots s_K
\]

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 \( p = q \) 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 blocks 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:

\[
x^{(1)}_1 \ldots x^{(1)}_p, \quad x^{(2)}_1 \ldots x^{(2)}_p
\]
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 $h_{p_1}'$ and $h_{p_2}'$ such that $h_{p_1}'\approx V h_{p_2}'$, and the two sequences of (distinct) variables corresponding to the groups of write blocks in $h_{p_1}'$ and $h_{p_2}'$:

$$y_1^{(1)} \ldots y_{p_1}^{(1)} \text{ and } y_1^{(2)} \ldots y_{p_2}^{(2)}$$

are the same for $h_{p_1}'$ and $h_{p_2}'$, i.e. $y_k^{(1)} = y_k^{(2)}$ ($1 \leq k \leq p_2$).

The only difference between $h_{p_1}'$ and $h_{p_2}'$ is the order in which write blocks occur within a write block grouping corresponding to a variable $x$. As noted above, $h_{p_1}'$ and $h_{p_2}'$ have the same last write blocks to $x$. It is clear from this and the construction of $h_{p_1}', h_{p_2}', h_{p_1}''$ and $h_{p_2}''$, that $h_{p_1}''$ and $h_{p_2}''$ have the same last write blocks to any given $x$. The other write blocks to this $x$ in $h_{p_1}''$ are just a permutation, $\pi_x$ say, of the remaining write blocks to $x$ in $h_{p_2}''$. By repeated use of Lemma 4 for each variable $x$ in turn we deduce that $h_{p_1}'' \approx V h_{p_2}''$.

To summarize,

$$h_1 \approx V h_1' \approx V h_{p_1}'' \approx V h_{p_2}'' \approx V h_2' \approx V h_2$$

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 $Vequiv(h_1, h_2) = TRUE$ 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 $tserzh(h) = TRUE$ can be proved for the term $h$ corresponding to $h$.

Proof $h$ tiered-serializable $\implies h$ equivalent to tiered-serial history $h_1$

$h_1$ tiered-serial, $h$ and $h_1$

$T$-equivalent and $V$-equivalent

$h_1$ tiered-serial, $h$ and $h_1$

$T$-equivalent and $h \approx V h_1$

(by Lemma 5)

$tserzh(h_1, n) = TRUE$ for some $n,$

$Tequiv(h_1, n) = TRUE,$

$Vequiv(h_1, h) = TRUE,$

for terms $h$, $h_1$ corresponding to $h_1$, respectively

(by (3) and Lemma 6)

$tserzh(h) = TRUE$ (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:

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

and the history $h_1$ of $\{T_1, T_2, T_3\}$ be $h_1 = r_1\tau_1r_1w_{11}w_{12}w_{13}w_{14}w_{15}w_{16}, r_2\tau_1r_2w_{21}w_{22}w_{23}w_{24}r_3w_{25}r_3w_{26}w_{27}w_{28}w_{29}$

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

$$Vequiv(w_{11}^1w_{11}^2w_{12}^1w_{12}^2w_{13}^1w_{13}^2w_{14}^1w_{14}^2w_{15}^1w_{15}^2w_{16}^1w_{16}^2) = TRUE$$

By commuting the write blocks corresponding to the single steps $w_{11}^1$ and $w_{12}^1$ (E33), (E23) and (E24)) in the second term in (9), we have that

$$Vequiv(w_{11}^1w_{12}^1w_{13}^1w_{12}^1w_{13}^2w_{14}^1w_{12}^2w_{13}^1w_{14}^2w_{15}^1w_{15}^2w_{16}^1w_{16}^2) = TRUE$$

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

$$Vequiv(w_{11}^1w_{12}^1w_{13}^1w_{12}^1w_{13}^2w_{14}^1w_{12}^2w_{13}^1w_{14}^2w_{15}^1w_{15}^2w_{16}^1w_{16}^2) = TRUE$$

By (9), (10) and (11) and transitivity of $Vequiv ((E22))$, $Vequiv(w_{11}^1w_{12}^1w_{13}^1w_{12}^1w_{13}^2w_{14}^1w_{12}^2w_{13}^1w_{14}^2w_{15}^1w_{15}^2w_{16}^1w_{16}^2) = TRUE$

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

$$Vequiv(h_2, h_1) = TRUE$$

where $h_2$ is the history $h_2 = r_1^1r_2^1r_3^1w_{11}^1w_{12}^1w_{13}^1w_{14}^1w_{15}^1w_{16}^1r_1^2r_2^1r_3^1r_2^2w_{21}^1w_{22}^1w_{23}^1w_{24}^1w_{25}^1w_{26}^1w_{27}^1w_{28}^1w_{29}^1$

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

$$Vequiv(h_2, h_1) = TRUE$$

where $h$ is the history $h = r_1^1w_{11}^1w_{12}^1w_{12}^2w_{12}^1r_3^1w_{12}^1w_{13}^1w_{14}^1w_{15}^1w_{16}^1r_1^2r_2^1r_3^1r_2^2w_{21}^1w_{22}^1w_{23}^1w_{24}^1w_{25}^1w_{26}^1w_{27}^1w_{28}^1w_{29}^1$

By (13), (14) and transitivity of $Vequiv ((E22))$,

$$Vequiv(h_1, h) = TRUE$$

By straightforward applications of (E6)–(E14), $h$ is clearly tiered-serial, i.e.

$$tserzh(h, 3) = TRUE$$

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

$$Vequiv(h_1, h) = TRUE$$

By (16), (17), (15) and (E34), we conclude that $tserzh(h_1)=TRUE$ 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 π-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