International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:4, No:3, 2010

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 commutativity-based forms of serializability have also been characterized in temporal logic [14] and [8]. However, there have been almost no algebraic or proof-rule characterizations of forms of serializability that are not commutativity-based. The specification in [13] is in a temporal logic without an evident axiomatization and although modest extensions beyond commutativity-based serializability are possible by the techniques of [10] and [8], no proof-theoretic method that uses inference rules has been formulated for general non-commutative 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].

Walter Hussak (corresponding author) is a lecturer in the Department of Computer Science, Loughborough University, UK (Email : W.Hussak@lboro.ac.uk, Tel. : +44(0)1509 222937, Fax: +44(0)1509 211586).
John Keane is a Professor in the School of Informatics, University of Manchester, UK (Email : John.Keane@manchester.ac.uk).

In this paper, we give an algebraic method for proving general serializablility. 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

$$\mathtt{r(i,x)} \text{ or } \mathtt{w(i,x)}$$

respectively. A more condensed notation will be used in places in Section IV where

$$r_i^x \text{ and } w_i^x$$

will replace `r(i,x)` and `w(i,x)`. The i subscript or x superscript will also be omitted occasionally when of no interest. A *transaction* $(T_i, \leq_i)$ 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 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

$$s_1 <_i s_2 \ \Rightarrow\ s_1 <_h s_2$$

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:

$$\mathtt{r(i,a)w(i,a)r(i,b)w(i,b)}$$

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:

$$\mathtt{r(1,a)w(1,a)r(2,a)w(2,a)r(2,b)w(2,b)r(1,b)w(1,b)}$$

B. Partitions and tiers

Let $T_\mathcal{P} = \{T_1, \ldots, T_n\}$ 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 $(1 \le i \le n)$. 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:

$$\mathtt{r(j,i,x)} \text{ or } \mathtt{w(j,i,x)}$$

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 (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

$$s_1 <_i s_3 \ \text{ and } \ s_4 <_i s_2 \tag{1}$$

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 ρ of $\{1, \ldots, K\}$ such that h' is

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

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

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

or

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

and *serializable* iff it is equivalent to a serial history. If the transactions are partitioned by a tiered partition $\mathcal{P}_t$, then a history of T is $\mathcal{P}_t$*-serial* or just *tiered-serial* (with the understanding that the tiers are as in $\mathcal{P}_t$) 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

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

or

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

Clearly, a serial history is tiered-serial. A history h is $\mathcal{P}_t$*-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:

$$\mathtt{r(i,x)w(i,x)r(i,y)w(i,y)}$$

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

$$\begin{aligned}&\mathtt{r(1,x)w(1,x)r(2,x)w(2,x)r(2,y)w(2,y)}\\ &\mathtt{r(1,y)r(3,x)w(1,y)w(3,x)r(3,y)w(3,y)}\end{aligned} \tag{2}$$

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 accesses to first flight and connecting flight information respectively constitute the tiers, i.e. each T_i is the tiered transaction:

$$\mathtt{r(1,i,x)w(1,i,x)r(2,i,y)w(2,i,y)}$$

and (2) becomes the tiered-serializable history:

$$\mathtt{r(1,1,x)w(1,1,x)r(1,2,x)w(1,2,x)r(2,2,y)w(2,2,y)}$$

$$\mathtt{r(2,1,y)r(1,3,x)w(2,1,y)w(1,3,x)r(2,3,y)w(2,3,y)}$$

equivalent to the tiered-serial history:

$$\mathtt{r(1,1,x)w(1,1,x)r(1,2,x)w(1,2,x)r(2,2,y)w(2,2,y)}$$

$$\mathtt{r(2,1,y)w(2,1,y)r(1,3,x)w(1,3,x)r(2,3,y)w(2,3,y)}$$

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

$$\mathtt{r(1,1,x)r(1,2,x)w(1,1,x)w(1,2,x)r(2,2,y)w(2,2,y)}$$

$$\mathtt{r(2,1,y)r(1,3,x)w(2,1,y)w(1,3,x)r(2,3,y)w(2,3,y)}$$

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 $\mathcal{P}_t$-serializability for tiered partitions $\mathcal{P}_t$. A similar definition of $\mathcal{P}$-serializability (see below) can be given for arbitrary partitions $\mathcal{P}$ 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 $\mathcal{P}$-serial histories for an arbitrary partition $\mathcal{P}$ is equal to the set of $\mathcal{P}_t$-serial histories for some tiered partition $\mathcal{P}_t$. As such, a required data consistency condition that is defined in terms of $\mathcal{P}$-serializability for some partition $\mathcal{P}$ can, with a little extra effort, be defined in terms of $\mathcal{P}_t$-serializability for some tiered partition $\mathcal{P}_t$.

A. $\mathcal{P}$-definability

Let $T = \{T_1, \ldots, T_n\}$ be a transaction system and let $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ 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

$$s_1 <_h s_2 \text{ for all } s_1 \in P_1,\ s_2 \in P_2$$

or

$$s_2 <_h s_1 \text{ for all } s_1 \in P_1,\ s_2 \in P_2$$

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

$$s_1 \le_h s_2 \le_h \cdots \le_h s_{2i-1} \le_h s_{2i} \le_h \cdots \le_h s_{2m-1} \le_h s_{2m}$$

such that:

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

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

The subsequence of h comprising the steps in $\{s : s_{2i-1} \le_h s \le_h s_{2i}\}$ is called the *i-th slice* of h $(1 \le i \le m)$. 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 $\sim_i$ on a transaction $T_i \in T$ to be such that $s_1 \sim_i s_2$ iff, for all $h \in \mathcal{H}$,

$$s_1 \leq_h s \leq_h s_2 \ \vee s_2 \leq_h s \leq_h s_1 \Rightarrow s \in T_i$$

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 $\sim_i$ is an equivalence relation on T_i. It is trivially reflexive and symmetric. Suppose that $s_1 \sim_i s_2$ and $s_2 \sim_i s_3$. Let $h \in H$ and s be such that $s_1 \leq_h s \leq_h s_3$. If $s_2 \leq_h s$, then $s_2 \leq_h s \leq_h s_3$ and so $s \in T_i$ as $s_2 \sim_i s_3$. If $s <_h s_2$, then $s_1 \leq_h s <_h s_2$ and so $s \in T_i$ as $s_1 \sim_i s_2$. By a symmetric argument $s_3 \leq_h s \leq_h s_1 \Rightarrow s_i \in T_i$. It follows that $\sim_i$ 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 $s_1 \not\sim_i s_3$, 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 $s_1 \not\sim_i s_2$ contradicting the fact that s_1 and s_2 belong to the same equivalence class T_{iq}. On the other hand, if $s_2 <_h s$ then

$$s_4 <_h s_2 <_h s \leq_h s_3$$

as $s_4 <_i s_2$ implies $s_4 <_h s_2$. This means that $s_4 \not\sim_i s_3$ 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, $\mathcal{P}_t$ say, of T.

Finally, it remains to show that $\mathcal{H}$ is *$\mathcal{P}_t$-definable*. Now, as $\mathcal{H}$ is $\mathcal{P}$-definable, it consists of the $\mathcal{P}$-serial histories of T. If s_1 and s_2 belong to the same $\mathcal{P}$-segment of T_i then, given a history $h \in \mathcal{H}$, as h is $\mathcal{P}$-serial, it 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_2 in h. Therefore, by the definition of $\sim_i$, $s_1 \sim_i s_2$ and so 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, $s_1 \sim_i s_2$. Therefore, the slices of each $h \in \mathcal{H}$ are unions of complete tiers. It follows that

$$\begin{array}{lll} h \in \mathcal{H} & \Leftrightarrow & h \text{ is } \mathcal{P}\text{-serial} \\ & \Leftrightarrow & \text{the slices of } h \text{ are a union of complete} \\ & & \mathcal{P}\text{-segments (by Lemma 1)} \\ & \Leftrightarrow & \text{the slices of } h \text{ are a union of complete tiers} \\ & \Leftrightarrow & h \text{ is } \mathcal{P}_t\text{-serial (by Lemma 1)} \end{array}$$

■

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

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

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

$$r_i^x w_i^x w_j^x r_k^y w_k^x \quad (x \neq y, i \neq j, j \neq k, i \neq k)$$

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

$$r_i^x w_i^x w_j^x r_k^x w_k^x \quad (x \neq y, i \neq j, j \neq k, i \neq k)$$

as r_k^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

$$r_i^x w_j^x r_k^x w_i^x w_k^x$$

where the write blocks w_i^x and $w_j^x r_k^x$ 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,i,x)` and `w(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 `1`, `succ` and `isequal`. Histories are formed by use of the associative infix operator . to concatenate other histories. For example, the following expression:

`r(1,succ(1),succ(succ(1))).w(1,succ(1),succ(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 `tier`, `trans` and `tser` have the following meanings: `tier(h,j,i)` asserts that `h` is a `j`-labelled tier of an `i`-labelled transaction, `trans(h,m,i)` asserts that `h` is an `i`-labelled transaction with `m` tiers, and `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 . 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 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,i1,i2,...,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)   _isequal_  : N,N    -> B
(O7)   _=_        : 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
      IF tier(h,j,i) = TRUE
(E9)  tier(h.w(j,i,x),j,i) = TRUE
      IF tier(h,j,i) = TRUE

(E10) trans(h,1,i) = TRUE
      IF tier(h,1,i) = TRUE
(E11) trans(h1.h2,succ(m),i) = TRUE
      IF trans(h1,m,i) = TRUE,
         tier(h2,succ(m),i) = TRUE

(E12) tser(h,1) = TRUE
      IF trans(h,m,1) = TRUE
(E13) tser(h1.h2,succ(n)) = TRUE
      IF tser(h1,n) = TRUE,
         trans(h2,m,succ(n)) = TRUE
(E14) tser(h1.h2.h4.h3.h5.h6,n) = TRUE
      IF tser(h1.h2.h3.h4.h5.h6,n) = TRUE,
         tier(h2,j2,i2) = TRUE,
         tier(h3,j3,i3) = TRUE,
         tier(h4,j4,i4) = TRUE,
         tier(h5,j5,i5) = TRUE,
         j2 isequal j3 = FALSE OR
         i2 isequal i3 = FALSE,
         j4 isequal j5 = FALSE OR
         i4 isequal i5 = FALSE,
         i3 isequal i4 = FALSE

(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(h.h1,h.h2)
      IF Tequiv(h1,h2) = TRUE
(E19) Tequiv(h1.h,h2.h)
      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(h.h1,h.h2)
      IF Vequiv(h1,h2) = TRUE
(E24) Vequiv(h1.h,h2.h)
      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) Vequiv(r(j1,i1,x).r(j2,i2,y),
      r(j2,i2,y).r(j1,i1,x)) = TRUE
(E31) Vequiv(r(j1,i1,x).w(j2,i2,y),
      w(j2,i2,y).r(j1,i1,x)) = TRUE
      IF x isequal y = FALSE
(E32) Vequiv(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
```

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, h) = TRUE and Vequiv(h1, h) = TRUE by (E34) for some history h1. To this end, we first define the two new equivalences for histories, namely 'T-equivalence' and 'V-equivalence', which together amount to history equivalence as defined in II.C, 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* iff 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.

$$h_1, h_2 \text{ are T-equivalent} \Leftrightarrow \mathtt{Tequiv(h1,h2)} = \mathtt{TRUE} \tag{3}$$

where `Tequiv(h1,h2)=TRUE` can be proved for the terms `h1` and `h2` 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* iff they have the same *sees* function and the same last write steps to each variable x. It is easy to see that h_1 and h_2 are equivalent (in the sense of II.C) 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

$$w^{x_1}r^{x_1}\dots r^{x_1}w^{x_2}r^{x_2}\dots r^{x_2}\dots w^{x_p}[r^{x_p}\dots r^{x_p}] \quad (p \geq 1)$$

where the square brackets indicate that there may or may not be read steps after the last write step w^{x_p}. The relation $\approx^V$ is the transitive closure of the reflexive and symmetric relation $\approx^{V^0}$ on histories which has $h_1 \approx^{V^0} h_2$ iff any of the following is the case:

$$\begin{aligned} h_1 &= s_1\dots s_{k-1}r_i^x r_j^y s_{k+2}\dots s_K, \\ h_2 &= s_1\dots s_{k-1}r_j^y r_i^x s_{k+2}\dots s_K, \end{aligned} \tag{4}$$

$$\begin{aligned} h_1 &= s_1\dots s_{k-1}r_i^x w_j^y s_{k+2}\dots s_K, \\ h_2 &= s_1\dots s_{k-1}w_j^y r_i^x s_{k+2}\dots s_K, \quad (x \neq y) \end{aligned} \tag{5}$$

$$\begin{aligned} h_1 &= s_1\dots s_{k-1}w_i^x w_j^y s_{k+2}\dots s_K, \\ h_2 &= s_1\dots s_{k-1}w_j^y w_i^x s_{k+2}\dots s_K, \quad (x \neq y) \end{aligned} \tag{6}$$

$$\begin{aligned} h_1 &= s_1\dots s_k w_{i_{11}}^x r_{i_{12}}^x \dots r_{i_{1k_1}}^x w_{i_{21}}^x r_{i_{22}}^x \dots r_{i_{2k_2}}^x w_j^x s_l \dots s_K, \\ h_2 &= s_1\dots s_k w_{i_{21}}^x r_{i_{22}}^x \dots r_{i_{2k_2}}^x w_{i_{11}}^x r_{i_{12}}^x \dots r_{i_{1k_1}}^x w_j^x s_l \dots s_K \end{aligned} \tag{7}$$

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_j^x to x as in (7). As (4), (5), (6) and (7) correspond to `(E30)`, `(E31)`, `(E32)` and `(E33)` respectively

$$h_1 \approx^V h_2 \Leftrightarrow \mathtt{Vequiv(h1,h2)} = \mathtt{TRUE} \tag{8}$$

for the terms `h1` and `h2` representing 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

$$h = s_1\dots s_k \underbrace{w^x r^x \dots r^x}_{\text{w.block } 1} \dots \underbrace{w^x r^x \dots r^x}_{\text{w.block } n} s_{l-1}s_l\dots s_K$$

where s_{l-1} is a write step to a variable x and n write blocks to x precede s_{l-1} in h. Then, given a permutation π of $\{1,\dots,n\}$, for

$$h' = s_1\dots s_k \underbrace{w^x r^x \dots r^x}_{\text{w.block } \pi(1)} \dots \underbrace{w^x r^x \dots r^x}_{\text{w.block } \pi(n)} s_{l-1}s_l\dots s_K$$

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 *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 r_i^x and w_j^y 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

$$h_1 \approx^V h_1', \; h_2 \approx^V h_2'$$

say

$$\begin{aligned} h_1' &= w^{x_1}r^{x_1}\dots r^{x_1}w^{x_2}r^{x_2}\dots r^{x_2}\dots w^{x_p}[r^{x_p}\dots r^{x_p}], \\ h_2' &= w^{y_1}r^{y_1}\dots r^{y_1}w^{y_2}r^{y_2}\dots r^{y_2}\dots w^{y_q}[r^{y_q}\dots r^{y_q}] \end{aligned}$$

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:

$$x_1^{(1)}\dots x_{p_g}^{(1)} \text{ and } x_1^{(2)}\dots x_{p_g}^{(2)}$$

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 $h_1''{\approx^V}h_1'''$, $h_2''{\approx^V}h_2'''$ and the two sequences of (distinct) variables corresponding to the groups of write blocks in h_1''' and h_2''':

$$y_1^{(1)} \dots y_{p_g}^{(1)} \text{ and } y_1^{(2)} \dots y_{p_g}^{(2)}$$

are the same for h_1''' and h_2''', i.e. $y_k^{(1)}= y_k^{(2)}$ $(1 \le k \le p_g)$.

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''' are just a permutation, π_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 $h_1'''{\approx^V}h_2'''$.

To summarize,

$$h_1 \approx^V h_1' \approx^V h_1'' \approx^V h_1''' \approx^V h_2''' \approx^V h_2'' \approx^V h_2' \approx^V h_2$$

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 $\mathtt{Vequiv(h1,h2)} = \mathtt{TRUE}$ can be proved for the terms $\mathtt{h1}$ and $\mathtt{h2}$ 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 `tserz(h)=true` can be proved for the term `h` corresponding to h.

Proof

$$\begin{array}{lll}
h \text{ tiered-serializable} & \Leftrightarrow & h \text{ equivalent to} \\
& & \text{tiered-serial history } h_1 \\
& \Leftrightarrow & h_1 \text{ tiered-serial, } h \text{ and } h_1 \\
& & \text{T-equivalent and V-equivalent} \\
& \Leftrightarrow & h_1 \text{ tiered-serial, } h \text{ and } h_1 \\
& & \text{T-equivalent and } h \approx^V h_1 \\
& & \text{(by Lemma 5)} \\
& \Leftrightarrow & \mathtt{tser(h1,n)} = \mathtt{TRUE} \text{ for some } \mathtt{n}, \\
& & \mathtt{Tequiv(h1,n)} = \mathtt{TRUE}, \\
& & \mathtt{Vequiv(h1,h)} = \mathtt{TRUE}, \\
& & \text{for terms } \mathtt{h},\ \mathtt{h1} \text{ corresponding} \\
& & \text{to } h, h_1 \text{ respectively} \\
& & \text{(by (3) and Lemma 6)} \\
& \Leftrightarrow & \mathtt{tserz(h)} = \mathtt{TRUE} \text{ (by (E34))}
\end{array}$$

■

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 $(\mathtt{E1}) - (\mathtt{E34})$ of IV.A. For $i{=}1,2,3$, let the transaction T_i be given by:

`r(1,i,x)w(1,i,y)w(1,i,z)r(2,i,y)r(2,i,z)w(2,i,x)`

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

$$r_{11}^x r_{12}^x r_{13}^x \underline{w_{11}^y w_{12}^y w_{12}^z w_{11}^z w_{13}^y w_{13}^z} r_{21}^y r_{21}^z w_{21}^x r_{22}^y r_{22}^z w_{22}^x r_{23}^y r_{23}^z w_{23}^x$$

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

$$\mathtt{Vequiv}(w_{11}^y \underline{w_{11}^z w_{12}^y w_{12}^z w_{13}^y} w_{13}^z, w_{11}^y \underline{w_{12}^y w_{13}^y w_{11}^z w_{12}^z} w_{13}^z) = \mathtt{TRUE} \tag{9}$$

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

$$\mathtt{Vequiv}(w_{11}^y w_{12}^y w_{13}^y \underline{w_{11}^z w_{12}^z w_{13}^z}, w_{11}^y w_{12}^y w_{13}^y \underline{w_{12}^z w_{11}^z w_{13}^z}) = \mathtt{TRUE} \tag{10}$$

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

$$\mathtt{Vequiv}(w_{11}^y w_{12}^y \underline{w_{13}^y w_{12}^z w_{11}^z} w_{13}^z, w_{11}^y w_{12}^y \underline{w_{12}^z w_{11}^z w_{13}^y} w_{13}^z) = \mathtt{TRUE} \tag{11}$$

By (9), (10) and (11) and transitivity of `Vequiv` (`(E22)`),

$$\mathtt{Vequiv}(w_{11}^y w_{11}^z w_{12}^y w_{12}^z w_{13}^y w_{13}^z, w_{11}^y w_{12}^y w_{12}^z w_{11}^z w_{13}^y w_{13}^z) = \mathtt{TRUE} \tag{12}$$

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

$$\mathtt{Vequiv}(h_2, h_1) = \mathtt{TRUE} \tag{13}$$

where h_2 is the history $h_2 =$

$$r_{11}^x r_{12}^x r_{13}^x \underline{w_{11}^y w_{11}^z w_{12}^y w_{12}^z w_{13}^y w_{13}^z} r_{21}^y r_{21}^z w_{21}^x r_{22}^y r_{22}^z w_{22}^x r_{23}^y r_{23}^z w_{23}^x$$

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

$$\mathtt{Vequiv}(h, h_2) = \mathtt{TRUE} \tag{14}$$

where h is the history $h =$

$$\underline{r_{11}^x w_{11}^y w_{11}^z}\ \underline{r_{12}^x w_{12}^y w_{12}^z}\ \underline{r_{13}^x w_{13}^y w_{13}^z}\ \underline{r_{21}^y r_{21}^z w_{21}^x}\ \underline{r_{22}^y r_{22}^z w_{22}^x}\ \underline{r_{23}^y r_{23}^z w_{23}^x}$$

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

$$\mathtt{Vequiv}(h, h_1) = \mathtt{TRUE} \tag{15}$$

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

$$\mathtt{tser}(h, 3) = \mathtt{TRUE} \tag{16}$$

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

$$\mathtt{Tequiv}(h, h_1) = \mathtt{TRUE} \tag{17}$$

By (16), (17), (15) and `(E34)`, we conclude that `tserz(`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

[1] E. Astesiano, M. Broy and G. Regio, “Algebraic Specification of Concurrent Systems,” in *Algebraic Foundations of System Specification*, IFIP State-of-the-Art Reports, chapter 13, Springer, 1999.

[2] E. Astesiano, M. Bidoit, H. Kirchner, B. Krieg-Brückner, P. Mosses, D. Sannella, and A. Tarlecki, “CASL: The Common Algebraic Specification Language,” *Theoretical Computer Science*, vol. 286(2), pp. 153-196, 2002.

[3] A.P. Black, V. Cremet, R. Guerraoui and M. Odersky, “An Equational Theory for Transactions,” *Proc. Foundations of Software Technology and Theoretical Computer Science 2003*, LNCS vol. 2914, Springer-Verlag, pp. 38-49, 2003.

[4] C.J. Date, *An Introduction to Database Systems*, Addison Wesley, 2004.

[5] H. Garcia-Molina and B. Kogan, “Achieving high availability in distributed databases,” *IEEE Transactions on Software Engineering*, vol. 14(7), 1988.

[6] J.A. Goguen, J.W. Thatcher and E.G. Wagner, “An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types,” in *Current Trends in Programming Methodology IV*, R.T. Yeh (ed.), pp. 68-95 Prentice-Hall, 1978.

[7] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[8] W. Hussak, “Serializable Histories in Quantified Propositional Temporal Logic,” *International Journal of Computer Mathematics*, vol. 81(10), pp. 1203-1211, 2004.

[9] W. Hussak and J.A. Keane, “Concurrency Control of Tiered Flat Transactions,” *Proc. 13th British National Conference on Databases*, LNCS vol. 940, Springer-Verlag, pp. 172-182, 1995.

[10] S. Katz and D. Peled, “Defining conditional independence using collapses,” *Theoretical Computer Science*, vol. 101, pp. 337-359, 1992.

[11] H. Korth and G. Speegle, “Formal Model of Correctness Without Serializability,” *Proc. ACM SIGMOD*, pp. 379-388, 1988.

[12] C. Papadimitriou, *The Theory of Database Concurrency Control*, Computer Science Press, 1986.

[13] D. Peled, S. Katz, and A. Pnueli, “Specifying and proving serializability in temporal logic,” *Proceedings LICS 1991*, pp. 232-245, IEEE Computer Society Press, 1991.

[14] D. Peled and A. Pnueli, “Proving partial order properties,” *Theoretical Computer Science*, vol. 126, pp. 143-182, 1994.

[15] D. Sangiorgi and D. Walker, *The π-calculus: a Theory of Mobile Processes*, Cambridge University Press, 2001.

[16] L. Sha, J.P. Lehoczky and E.D. Jensen, “Modular Concurrency Control and Failure Recovery,” *IEEE Transactions on Computers*, vol. 37(2), pp. 146-159, 1988.

[17] K. Vidyasankar, “Generalized Theory of Serializability,” *Acta Informatica*, vol. 24, pp. 105-119, 1987.

[18] P. Wolper, “Temporal Logic Can Be More Expressive,” *Information and Control*, vol. 56, pp. 72-99, 1983.