A Serializability Condition for Multi-step Transactions Accessing Ordered Data

Rafat Alshorman, Walter Hussak

Abstract—In mobile environments, unspecified numbers of transactions arrive in continuous streams. To prove correctness of their concurrent execution a method of modelling an infinite number of transactions is needed. Standard database techniques model fixed finite schedules of transactions. Later, techniques based on temporal logic have been proposed as suitable for modelling infinite schedules. The drawback of these techniques is that proving the basic serializability correctness condition is impractical, as encoding (the absence of) conflict cyclicity within large sets of transactions results in prohibitively large temporal logic formulae. In this paper, we show that, under certain common assumptions on the graph structure of data items accessed by the transactions, conflict cyclicity need only be checked within all possible pairs of transactions. This results in formulae of considerably reduced size in any temporal-logic-based approach to proving serializability, and scales to arbitrary numbers of transactions.

Keywords—multi-step transactions, serializability, directed graph.

I. INTRODUCTION

MOBILE computing and the World Wide Web (WWW) have produced huge numbers of simultaneous users executing transactions in transaction processing systems. The scenario is one of unbounded numbers of transactions incoming and outgoing to databases in continuous streams and accessing common data items [1]. It has been noted [2], [3], [5] that modelling such systems, from the point of view of ascertaining such properties as correctness and absence of starvation, requires modelling infinite rather than the fixed finite schedules of classical database theory [4]. Infinite schedules can be modelled in temporal logic [2], [3], [5]. However, there is a significant drawback to modelling the main serializability correctness condition.

The problem with specifying serializability is with specifying the existence of cycles in the infinite conflict graph of a schedule or ‘history’ h. There are n! ways that a cycle can occur between n transactions and an encoding into temporal logic results in a formula whose length is of order a factorial of the number of propositions that represent different active transactions at any point in time. In the case of a scheduler dealing with at most n active transactions any at point in time, exhaustive proofs of serializability are beyond even the most powerful model checkers available for any realistic value of n. The work [5] gives an encoding of serializability that is polynomial in such n, but uses a temporal logic of non-elementary computational complexity. In [2], a polynomial encoding into plain LTL is achieved for 2-step transactions. A much simpler encoding of serializability, that is more amenable to model checking, is given for multi-step transactions in [8], and requires cycles only to be checked between pairs of transactions. However, the assumption there is that all transactions access the same set of data items. In the general case, where transactions access different sets of data items, it is not sufficient to check for cycles between pairs of transactions. Consider the case of 3 transactions T₁, T₂ and T₃ which access the sets of data items {x, y}, {y, z} and {x, z} respectively, where x, y and z are all different, and the history h₁ whose steps occur in the following order:

\[ h₁ = r_3(x)w_3(x)r_1(x)w₁(x)r₁(y) \]
\[ w₁(y)r₂(y)w₂(y)r₂(z)w₂(z)r₃(z)w₃(z) \]

Here, for example, r₃(x) denotes transaction T₃ reading x, and w₂(y) transaction T₂ writing to x. History h₁ has the following cycle of conflicts:

\[ T₃ → T₁ → T₂ → T₃ \]

(and is therefore not serializable). However, as can be checked, there is no cycle between two transactions. In fact, as the history h₂ below shows, the absence of a cycle between any n − 1 transactions does not guarantee the absence of a cycle between n transactions:

\[ h₂ = r_n(x₁)w_n(x₁)r₁(x₁)w₁(x₁)r₁(x₂)w₁(x₂) \ldots \]
\[ \ldots r₁(x)w₁(x₁)r₁(x₁+1)w₁(x₁+1) \ldots \]
\[ \ldots rₙ₋₁(xₙ₋₁)wₙ₋₁(xₙ₋₁)rₙ₋₁(xₙ₋₁)wₙ₋₁(xₙ)wₙ(xₙ) \]

Here,

\[ T₁, \ldots, T_i, \ldots, Tₙ \]

access sets of data items

\[ \{x₁, x₂\}, \ldots, \{xᵢ, xᵢ₊₁\}, \ldots, \{xₙ, x₁\} \]

where x₁, . . . , xₙ are distinct. History h₂ has the cycle

\[ Tₙ → T₁ → \ldots → Tₙ₋₁ → Tₙ \]

but no cycle of fewer than n transactions - if Tᵢ (1 ≤ i ≤ n) is removed from the history, the (acyclic) order of conflicts is:

\[ Tᵢ₊₁ → \ldots → Tₙ → T₁ → \ldots → Tᵢ₋₁ \]

We notice that in the examples h₁ and h₂ above, it is impossible to define an order on the data items, so that each transaction only accesses sets of contiguous data items. In this paper, we show that if such an order exists, then
to check for conflict cyclicity within sets of transactions, it is sufficient to check for conflict cyclicity between pairs of transactions. This results in a test for serializability of concurrent multi-step transactions, similar to that in [8], but for an entirely different class of transactions where the sets of data items accessed by transactions need not be the same. This paper is structured as follows. First of all, in Section II, we give a model for infinite numbers of concurrent multi-step transactions accessing ordered data. In Section III, we prove that, in this model, a cycle exists in the (infinite) conflict graph if and only if a cycle exists between two transactions. A formal condition for serializability, that can be encoded into temporal logic, is given in Section IV. Applications that satisfy this condition for serializability, that can be encoded into temporal model are discussed in Section V and concluding remarks are given in Section VI.

II. CONCURRENT MULTI-STEP TRANSACTIONS MODEL

A. Histories

We shall denote the set of multi-step transactions by \( T = \{ T_i : i \in \mathbb{N}_1 \} \), where \( \mathbb{N}_1 \) is the set of positive integers. A history (or schedule) \( h \) for the set \( T \) is an interleaved sequence of all the read and write steps of all the transactions in \( T \) such that the subsequence of \( h \) comprising the steps of \( T_i \) is exactly the sequence of steps of \( T_i \) occurring in the order that they do in \( T_i \). We write \( s_i <_h s_j \) if step \( s_i \) of \( T_i \) occurs before step \( s_j \) of \( T_j \) in \( h \). A read (respectively write) step, of transaction \( i \) on data item \( x \), is denoted \( r_i(x) \) (respectively \( w_i(x) \)). In the next definition, we define formally an order on the set of the data items from which data is accessed by transactions.

**Definition 1.** Let \( D = \{ x_1, x_2, \ldots, x_m \} \) be an irreflexively totally ordered, by \( <_D \) say, set of data items such that

\[
x_1 <_D \ldots <_D x_m
\]

and \( T = \{ T_i : i \in \mathbb{N}_1 \} \) be the set of transactions participating in history \( h \). Denote by \( D_i \), the totally ordered set of data items accessed in turn by transaction \( T_i \) assumed to be of the form

\[
D_i = \{ x_a, x_{a+1}, \ldots, x_{b-1}, x_b \}
\]

where \( 1 \leq a \leq b \leq m \) and \( D_i \subseteq D \). For the remainder of this paper, if a set of data items \( D' \subseteq D \) is denoted by \( \{ x_a, \ldots, x_b \} \), this will mean that \( x_a <_D \ldots <_D x_b \). We shall denote the case of transaction \( T_i \) preceding transaction \( T_j \) in accessing data items \( x_p, \ldots, x_s \) in both read and write operations over history \( h \), as \( T_i <_h x_p, \ldots, x_s \ T_j \), where \( x_p, x_{p+1}, \ldots, x_s \in D_i \cap D_j \).

For example, assume that \( D_i \) and \( D_{i'} \) are:

\[
D_i = \{ x_1, x_2, x_3 \}
\]

\[
D_{i'} = \{ x_2, x_3, x_4, x_5 \}
\]

where \( x_1 <_D x_2 <_D x_3 <_D x_4 <_D x_5 \). Then, in the history

\[
h = \ldots r_1(x_2)w_1(x_3)\ldots r_1(x_1)w_1(x_3)\ldots r_1(x_2)w_1(x_3)\ldots r_1(x_3)w_1(x_3)\ldots \]

we have that \( T_i <_h x_2, x_3 \ T_{i'} \).

B. Serializability

A serial history is a history in which all operations of any transaction \( T_i \) are executed consecutively in the history. Otherwise, the history is called nonserial. A ‘serializable’ history is a history that is ‘equivalent’ to some serial history of the same transactions. Various notions of equivalence have been defined as in, for example, [4] and [2]. In this paper, we shall adopt the common ‘conflict equivalence’. Two histories are conflict equivalent if the order of any two ‘conflicting’ operations is the same in both histories. Two operations conflict if they belong to different transactions, access the same data item, and at least one of them is a write.

**Definition 2.** Histories \( h_1 \) and \( h_2 \) of \( T = \{ T_i : i \in \mathbb{N}_1 \} \) are equivalent, written as \( h_1 \sim h_2 \), iff for all \( i, i' \geq 1, i \neq i' \), and for all \( x \in D_i \cap D_{i'} \),

1) if \( r_i(x) <_{h_1} w_{i'}(x) \), then \( r_i(x) <_{h_2} w_{i'}(x) \) and
2) if \( w_i(x) <_{h_1} w_{i'}(x) \), then \( w_i(x) <_{h_2} w_{i'}(x) \)

**Definition 3.** A history \( h \) of \( T = \{ T_i : i \in \mathbb{N}_1 \} \) is serializable iff there is a serial history \( h_S \) of \( T \) of the form, for each \( i \in \mathbb{N}_1 \),

\[
h_S = \ldots r_i(x)w_i(y)\ldots
\]

only (all) steps of \( T_i \)

such that \( h \sim h_S \).

C. Conflict graphs

‘Conflict graphs’ are widely used for testing conflict serializability of finite histories of transactions in polynomial time [4], [6], [7]. Definition 4, below, explains how we build the conflict graph for a history. Theorem 5 states that acyclicity of conflict graphs corresponds to serializability in the case of infinite histories.

**Definition 4.** A directed graph is a pair \( G = (V, A) \), where \( V \) is a set of elements called nodes, denoted nodes \((G)\), and \( A \subseteq V \times V \) is a set of elements called arcs, denoted arcs \((G)\). A walk in a directed graph \( G = (V, A) \) is a sequence of nodes \((v_1, v_2, \ldots, v_n)\) such that \((v_i, v_{i+1}) \in A \) for \( i = 1, \ldots, n - 1 \). A walk with no nodes repeated is called a path; it is a cycle when only the first and last node coincide. For each history \( h \), there is a directed graph \( G(h) \) called the conflict graph of \( h \). This graph has the transactions of \( h \) as its nodes, and contains an arc \((T_i, T_{i'})\), where \( T_i \) and \( T_{i'} \) are distinct transactions of \( h \), whenever there is a step of \( T_i \) which conflicts with a subsequent (in \( h \)) step of \( T_{i'} \).

**Theorem 5.** A history \( h \) of an infinite number of multi-step transactions \( T = \{ T_i : i \in \mathbb{N}_1 \} \), accessing data items in some finite set \( D \), is serializable iff its conflict graph \( G(h) \) is acyclic.

**Proof:** This theorem is proved as Theorem 3.2 in [8].

III. CYCLE REDUCTION IN CONFLICT GRAPHS

In this section, we prove a succession of properties of infinite histories culminating in Theorem 11. Theorem 11 is the main result and is used to give a condition for serializability in Section IV that can be encoded efficiently into common
temporal logics such as CTL (Computational Tree Logic) or LTL (Linear-time Temporal Logic).

The first two lemmas, Lemma 6 and Lemma 7, give basic properties of the order \( <_h \) from Definition 1.

**Lemma 6.** Let \( h \) be a history with no cycle of length 2 in its conflict graph, and suppose that \( x \in D \) and transactions \( T_i \) and \( T_j \) are such that \( x \in D_i \cap D_j \). Then,

\[
either \ T_i <_h T_j \or T_j <_h T_i
\]

**Proof:** By the definition of \( <_h \) in Definition 1, the only way (1) can fail is if two reads \( r_i(x) \) and \( r_j(x) \) occur before the two writes \( w_i(x) \) and \( w_j(x) \). But, then the conflict graph would have the cycle \((T_i, T_j, T_k, T_i)\) of length 2.

**Lemma 7.** Let \( h \) be a history with no cycle of length 2 in its conflict graph \( G(h) \), \( T_i <_h x a \ T_j \), and suppose that \( x_a \) lies between \( x_a \) and \( x_a \), i.e. \( x_a <_D x_a <_D x_a \). Then, \( T_i <_h x \ T_j \), (see Figure 1).

**Proof:** As \( x_a, \ a \in D \cap D_j \), and as, by Definition 1, \( D_i \), \( D_j \), \( D_j \), contain contiguous elements, we have that \( x_a \in D_i \cap D_j \). If \( T_i <_h x \ T_j \) does not hold then, by Lemma 6, we must have that \( T_j <_h x \ T_i \). This implies that there is an arc in \( G(h) \) such that \((T_i, T_j)\). As \( T_i <_h x \ T_j \), there is another arc \((T_i, T_j)\) in \( G(h) \) thereby completing a cycle of length 2 and contradicting the assumption that \( G(h) \) has no cycle of length 2.

Fig. 1. The conflict graph \( G(h) \) for Lemma 7.

For the next property Lemma 8 below, consider the case of three transactions \( T_i \), \( T_j \), and \( T_k \), accessing data items such that the data items accessed by \( T_j \) straddle those accessed by \( T_i \) and \( T_k \) as follows:

\[
D = \{x_1, \ldots, \ x_n, \ x_{a_1}, \ldots, \ x_{a_1}, \ldots, \ x_{a_1}, \ldots, \ x_{a_2}, \ldots, \ x_{a_2}, \ldots, \ x_{a_2}, \ldots, \}
\]

We claim that \( T_i \), \( T_j \), and \( T_k \) cannot be part of a cycle in \( G(h) \) of length \( n \) where \( n > 2 \) if there is no cycle of length 2. For example, assume that there are 4 ordered sets of data items \( D_i \supseteq \{x_a, x_b\} \), \( D_j \supseteq \{x_a, x_b\} \), \( D_k \supseteq \{x_a, x_b\} \) and \( D_l \supseteq \{x_a, x_b\} \) accessed by corresponding transactions \( T_j, T_k \), \( T_i \), \( T_i \), \( T_i \), respectively, which form a cycle as in Figure 2 below. Since \( D_i \) contains \( x_a, x_b \) it follows, by Definition 1, that \( x_b \) and \( x_a \) are also in \( D_i \), i.e.,

\[
D_i \supseteq \{x_a, x_b, x_a, x_b\}.
\]

Thus, \( x_a \in D_i \cap D_i \) and so, by Lemma 6, either \( T_i <_h x_a T_i \) or \( T_i <_h x_a T_i \). If \( T_j <_h x_a T_i \) then, by transitivity of \( <_h \), we can reduce the cycle, as in Figure 3(a). But, if \( T_i <_h x_a T_i \) then, we have a cycle of length two; see Figure 3(b). Now, for the case of Figure 3(a), from above we have that \( x_b \in D_i \cap D_i \). Therefore, by Lemma 6, either \( T_j <_h x_a T_i \) or \( T_i <_h x_a T_i \). If \( T_i <_h x_a T_i \) then, by transitivity, we have a cycle of length two; see Figure 4(b). But, if \( T_i <_h x_a T_i \), then we also have a cycle of length two, see Figure 4(a). Below, Lemma 8 shows that in fact our claim is true for \( n \geq 3 \) number of transactions. Lemma 8 will be used to prove Lemma 9.

**Lemma 8.** Suppose that \( D_i \), \( D_j \), and \( D_k \) are the sets of data items accessed by \( T_i \), \( T_j \), and \( T_k \), respectively, and are of the form

\[
D_i = \{x_{a_1}, x_{a_1}, \ldots, x_{a_1}, \ldots, x_{a_2}, \ldots, \}
\]

\[
D_j = \{x_{a_1}, x_{a_1}, \ldots, x_{a_1}, \ldots, x_{a_2}, \ldots, x_{a_2}, \ldots, \}
\]

Fig. 3. We can either reduce the cycle or make a cycle of length two.

Fig. 4. We have a cycle of length two if either \( T_i <_h x_a T_i \) or \( T_i <_h x_a T_i \)

(a) If \( T_j <_h x_a T_i \) (b) If \( T_i <_h x_a T_i \)
$D_k = \{ \ldots, x_{b_i}', x_{b_i}, \ldots, x_{b_j}' \}$

where, for all $x_u \in \{ x_{a_i} : 1 \leq i \leq l \}$ and $x_{v'} \in \{ x_{a_j} : 1 \leq g \leq u \}$, $x_{v'} <_D x_u$, $x_{v'} \notin D_j$, $T_i <_{h^v} T_j$ and $T_i <_{h^v} T_k$.

Then, there is no cycle in $G(h)$ of minimum length $n$, where $n > 2$ is of the form:

\[(T_i, T_j), (T_j, T_k), \ldots, (T_i, T_{i+1}), \ldots, (T_s, T_i)\]

and $T_s <_{h^v} T_i$.

\[\text{Proof:}\] Assume, on the contrary, that we have a cycle of minimum length $n > 2$,

\[(T_i, T_j), (T_j, T_k), (T_k, T_2), \ldots, (T_s, T_i)\] \hspace{1cm} (s $\geq 0$) \hspace{1cm} (2)

(where $s = 0$ is the case $T_i = T_k$). Let $x_0 = x_{v'}, x_1, \ldots$, $x_s = x_a$, $x_{s+1} = x_{a'}$ be such that, putting $T_i = T_{s+1}, T_j = T_{s+2}$, and $T_k = T_0$, $x_{s+1}(= T_i) <_{h^v} T_{s+2}(= T_j)$, $T_{s+2}(= T_j) <_{h^v} T_0(= T_k)$, \ldots, $T_i <_{h^v} T_{s+1}, \ldots, T_s <_{h^v} T_{s+1}(= T_i)$.

We have that $x_c <_D x_{v'} <_D x_{v'}$, but, clearly, we cannot have $x_{v'} <_D x_{v'} >_D x_l >_D \ldots >_D x_l >_D \ldots >_D x_c >_D x_{v'}$.

Therefore, there is some $l$, with $0 \leq l \leq s - 1$, such that

\[x_l <_D x_{l+1} >_D x_l+2\] \hspace{1cm} (3)

There are two cases to consider corresponding to (3):

Case (i) $x_{l+1+2} <_D x_l <_D x_{l+1}$.

This case, as

\[x_{l+2} <_D x_l <_D x_{l+1},\]

and

\[T_i <_{h^v} T_{l+1} <_{h^v} T_{l+2} <_{h^v} T_{l+3}\]

we have that $x_{l+1}, x_{l+2}$ and therefore $x_l$ belong to $T_{(l+2)}$. As no cycle of length 2 exists between $T_{(l+1)}$ and $T_{(l+2)}$, we must have that $T_{(l+1)} <_{h^v} T_{(l+2)}$. Therefore, as $T_i <_{h^v} T_{l+1}$, by transitivity we have that $T_i <_{h^v} T_{l+2}$. This produces a cycle reduction of (2), which is a contradiction.

Case (ii) $x_l <_D x_{l+2} <_D x_{l+1}$.

This case, as

\[x_l <_D x_{l+2} <_D x_{l+1},\]

and

\[T_i <_{h^v} T_{l+1} <_{h^v} T_{l+2} <_{h^v} T_{l+3}\]

we have that $x_l, x_{l+1}$ and therefore $x_{l+2}$ belong to $T_{(l+2)}$. As no cycle of length 2 exists between $T_{(l+1)}$ and $T_{(l+2)}$, we must have that $T_{(l+1)} <_{h^v} T_{(l+2)}$. By transitivity, from $T_{(l+1)} <_{h^v} T_{(l+2)}$ and $T_{(l+2)} <_{h^v} T_{(l+3)}$, we get $T_{(l+1)} <_{h^v} T_{(l+2)}$ giving a cycle reduction. This contradiction completes the proof of the lemma.

Now, consider the case where $z <_D y <_D x$ and $T_1, T_2$ and $T_3$ access the following sets of data items:

\[D_1 = \{ x \}, D_2 = \{ y, x \} \quad \text{and} \quad D_3 = \{ z, y \}\]

Also, consider a history $h$, which contains $T_1, T_2$ and $T_3$, of the form:

\[h = \{ \ldots r_2(y) \ldots r_2(y) \ldots r_2(y) \ldots r_1(x) \ldots w_1(x) \ldots \ldots \ldots \}

The corresponding conflict graph $G(h)$, for the history $h$, is shown in Figure 5. This shows a situation that Lemma 9 can remove, i.e. Lemma 9 asserts that if we have cycle in $G(h)$ of length $n$, where $n > 2$, then, any three consecutive transactions $T_i, T_j$ and $T_k$, participating in $G(h)$, should contain $x_a$ and $x_{v'}$ such that $x_a <_D x_{v'}$, $T_i <_{h^v} T_j$ and $T_j <_{h^v} T_k$. We need such $x_a$ and $x_{v'}$ of Lemma 9 along with Lemma 10 to prove the main result Theorem 11 that reduces to cycles of length 2.

**Lemma 9.** Let $h$ be a history with a cycle in $G(h)$ of minimum length $n$, where $n > 2$, containing arcs $(T_i, T_j)$ and $(T_j, T_k)$, and $x_a, x_b$ such that $T_i <_{h^v} T_j$ and $T_j <_{h^v} T_k$. Then, there exist $x_{a'}, x_{v'}$ such that $x_{a'} <_D x_{v'}$ and $T_i <_{h^v} T_j$ and $T_j <_{h^v} T_k$.

**Proof:** Assume, on the contrary, that we have a cycle in $G(h)$ of length $n$, where $n > 2$, containing arcs $(T_i, T_j), (T_j, T_k)$ such that $T_i <_{h^v} T_j$ and $T_j <_{h^v} T_k$ for some $x_a, x_{v'}$, but that there are no $x_{a'}, x_{v'}$ such that $x_{a'} <_D x_{v'}$ and $T_i <_{h^v} T_j$ and $T_j <_{h^v} T_k$. This means that, for all $x_{v'} \in D_i \cap D_j$, $T_i <_{h^v} T_j$ and, for all $x_{v'} \in D_j \cap D_k$, $T_j <_{h^v} T_k$. We have that $x_{a'} <_D x_{v'}$. Therefore, $D_i$ contains data items $x_a \in D_i \cap D_j = \{ x_{a1}, x_{a2}, \ldots, x_{a l} \}$, $D_k$ contains data items $x_{v'} \in D_j \cap D_k = \{ x_{v'1}, x_{v'2}, \ldots, x_{v' m} \}$, and $D_j$ contains all data items $x_{a1}, x_{a2}, \ldots, x_{a l}, x_{v'1}, x_{v'2}, \ldots, x_{v' m}$, i.e. $D_j$ is of the form:

\[D_j = \{ \ldots, x_{a1}, x_{a2}, \ldots, x_{a l}, x_{v'1}, x_{v'2}, \ldots, x_{v' m} \} \]

We show, from (4), that, in fact, $D_i$ and $D_k$ should be as follows:

\[D_i = \{ x_{a1}, x_{a2}, \ldots, x_{a l} \} \]

\[D_k = \{ x_{v'1}, x_{v'2}, \ldots, x_{v' m} \} \]

Firstly, assume that (5) does not hold, i.e we can find $x_c \in D_i$ such that $x_c <_D x_{a l}$ and

\[D_i = \{ x_{a1}, x_{a2}, \ldots, x_{a l} \}.\]

From (4), we can choose $x_c$ to be such that $x_c \in D_i$ and therefore $x_c \in D_i \cap D_j$. If $x_c$ is such that $T_i <_{h^v} T_j$ then,
With the arc 

This contradicts our assumption that 

Because then 

and a transaction 

leads to a contradiction. Since, we have a cycle, there is the hypothesis of this lemma. Thus, (6) must hold.

We shall now show that our main assumption that no exists such that 

leads to a contradiction. Since, we have a cycle, there is 

and a transaction such that 

We cannot choose such 

because then we could reduce the cycle as we would have 

and therefore 

and an arc 

Also, we cannot choose such 

and 

as we could reduce the cycle because then 

However, if 

and 

we have that 

and 

giving the conditions of Lemma 8. Application of Lemma 8 shows that 

and 

could not form part of a cycle. This contradiction completes the proof.

Lemma 10. If 

is a history with a cycle in 

where 

and there are 

such that 

Then, 

(see Figure 6). Put:

(8)

(see Figure 8), where 

is with respect to 

By Lemma 9, there exist 

and 

such that:

As 

, then by Lemma 7 and (7), 

Therefore, we should have either 

or 

. Now, if 

then we can reduce the cycle via giving a contradiction. But, if 

then this gives an arc 

which with the arc 

completes a cycle of length 2 which is also a contradiction.

In a similar fashion, we can prove that:

(10)

The contradiction completes the proof.

Theorem 11 shows that if we have a cycle in the conflict graph 

of length 

where 

then there is a cycle of length two in 

This result is the basis of the serializability condition for these kinds of transactions, given in Theorem 12, below.
Theorem 12. A history \( h \) of transactions \( T = \{ T_i : i \in \mathbb{N} \} \) is serializable iff for any two transactions \( T_i, T_j \in T \) \((i, j \geq 1, i \neq j)\), where \( T_i \) say, is such that, for all \( x \in D_i \cap D_j \), \( w_i(x) <_{h} r_{i'}(x) \).

**Proof:** Only if part

Assume that \( h \) is a history where the condition is not satisfied. One possibility is that there exists a data item \( x_1 \) such that \( r_{i'}(x_1) <_h w_1(x_1) \), giving an arc \((T_i, T_j)\) in \( G(h) \), and another different data item \( x_2 \) such that \( w_2(x_2) <_h r_{i'}(x_2) \), giving an arc \((T_i, T_j)\) in \( G(h) \) thereby completing a cycle. Hence, in this case, by Theorem 5, \( h \) is not serializable. The other possibility is that the condition is breached on a single data item, i.e. one of the following cases:

\[
\ldots r_i(x) \ldots r_{i'}(x) \ldots w_i(x) \ldots w_{i'}(x) \ldots \quad (11)
\]

\[
\ldots r_i(x) \ldots r_i(x) \ldots w_i(x) \ldots w_{i'}(x) \ldots \quad (12)
\]

\[
\ldots r_i(x) \ldots r_{i'}(x) \ldots w_i(x) \ldots w_{i'}(x) \ldots \quad (13)
\]

\[
\ldots r_{i'}(x) \ldots r_i(x) \ldots w_{i'}(x) \ldots w_i(x) \ldots \quad (14)
\]

In cases (11)-(14), it is clear that the conflict graphs of the histories are cyclic and, by Theorem 5, not serializable.

If part

Assume that the history \( h \) is not serializable. We show that the condition does not hold. To say that \( h \) is not serializable means, by Theorem 5, that there exists a cycle in \( G(h) \). From Theorem 11, \( G(h) \) has a cycle \((T_i, T_{i'}, (T_i, T_j))\), where \( i, i' \geq 1 \) and \( i \neq i' \). Then, \( h \) is one of the following forms:

\[
\ldots r_i(x) \ldots r_{i'}(x) \ldots w_i(x) \ldots w_{i'}(x) \ldots \quad (15)
\]

\[
\ldots r_i(x) \ldots r_i(x) \ldots w_i(x) \ldots w_{i'}(x) \ldots \quad (16)
\]

\[
\ldots r_i(x) \ldots r_{i'}(x) \ldots w_i(x) \ldots w_{i'}(x) \ldots \quad (17)
\]

\[
\ldots r_{i'}(x) \ldots r_i(x) \ldots w_{i'}(x) \ldots w_i(x) \ldots \quad (18)
\]

\[
\ldots r_i(x) \ldots w_i(y) \ldots w_i(x) \ldots \quad (19)
\]

\[
\ldots r_{i'}(x) \ldots w_i(y) \ldots w_{i'}(x) \ldots \quad (20)
\]

In (15)-(20) steps are underlined if they cause the condition to be breached. Cases (15)-(18) are when only one data item \( x \) causes a cycle, and cases (19) and (20) are when 2 data items \( x \) and \( y \) cause a cycle.

**Definition 13.** If \( T' \subseteq T \), then the projection of \( h \) to \( T' \), denoted \( h_{T'} \), is the history of \( T' \), obtained from \( h \), by deleting all steps of transactions not in \( T' \).

To explain how the serializability condition of Theorem 12 is used to verify whether a history \( h \) is serializable or not, we shall give the following example:

Let \( T_1, T_2, T_3 \) and \( T_4 \) be multi-step transactions as follows

\[
T_1 = r_1(x_2)w_1(x_2)r_1(x_3)w_1(x_3)r_1(x_4)w_1(x_4)
\]

\[
T_2 = r_2(x_1)w_2(x_1)r_2(x_2)w_2(x_2)
\]

\[
T_3 = r_3(x_2)w_3(x_2)r_3(x_3)w_3(x_3)r_3(x_4)w_3(x_4)r_3(x_5)w_3(x_5)
\]

\[
T_4 = r_4(x_2)w_4(x_2)r_4(x_3)w_4(x_3)
\]

Let \( D \) be the set of all data items as follows

\[
D = \{ x_1, x_2, x_3, x_4, x_5 \}
\]

Also, let \( h_{T'} \) be the history of \( T' = \{ T_i : 1 \leq i \leq 4 \} \), where \( T' \subseteq T \).

\[
h_{T'} = r_1(x_2)r_2(x_1)w_2(x_1)r_2(x_2)r_3(x_2)w_3(x_2)
\]

\[
w_2(x_2)r_4(x_2)r_3(x_2)r_3(x_3)w_3(x_3)
\]

\[
r_4(x_2)w_4(x_3)r_1(x_4)w_1(x_4)r_3(x_4)w_3(x_4)r_3(x_5)w_3(x_5)
\]

Firstly, we shall chop the history \( h_{T'} \) up into sets of histories each containing two different transactions. Then, we shall check whether the serializability condition is satisfied for each set, to see if the history \( h_{T'} \) is serializable. If \( h_{T'} \) is not serializable, the main history \( h \) will not be serializable.

Consider:

\[
h_{(T_1, T_2)} = r_1(x_2)r_2(x_1)w_1(x_2)r_2(x_2)w_2(x_2)r_1(x_3)
\]

\[
w_1(x_3)r_1(x_4)w_1(x_4)
\]

We notice, from \( h_{(T_1, T_2)} \), that \( D_1 \cap D_2 = \{ x_2 \} \). According to the serializability condition in Theorem 12, if \( h \) is serializable then, we should have either, for all \( x \in D_1 \cap D_2, w_1(x) <_h r_2(x) \) or, for all \( x \in D_1 \cap D_2, w_2(x) <_h r_1(x) \). From \( h_{(T_1, T_2)} \), we have \( w_1(x_2) <_h r_2(x_2) \). This means that \( T_1 \) and \( T_2 \) satisfy the condition. Next, consider:

\[
h_{(T_1, T_3)} = r_1(x_2)w_1(x_2)r_3(x_2)r_3(x_3)w_1(x_3)r_3(x_3)
\]

\[
w_3(x_3)r_1(x_4)w_1(x_4)r_3(x_4)w_3(x_4)r_3(x_3)r_3(x_5)w_3(x_5)
\]
From \( h(T_1, T_2) \), we notice that \( D_1 \cap D_3 = \{ x_2, x_3, x_4 \} \). Also, we have for all \( x \in D_1 \cap D_3 \), \( w_1(x) \prec h r_3(x) \). This means that \( T_1 \) and \( T_3 \) satisfy the condition. Next, consider:

\[
h(T_1, T_4) = r_1(x_2)w_1(x_2)r_4(x_2)w_4(x_2)r_1(x_3)w_1(x_3)r_4(x_3)w_4(x_3)r_1(x_4)w_1(x_4).
\]

In \( h(T_1, T_4) \), we have, for all \( x \in D_1 \cap D_4 = \{ x_2, x_3 \} \), \( w_1(x) \prec h r_3(x) \). This means that \( T_1 \) and \( T_4 \) satisfy the condition. Next, consider:

\[
h(T_2, T_3) = r_2(x_1)w_2(x_1)r_3(x_2)w_3(x_2)r_2(x_3)w_2(x_3)r_3(x_3)w_3(x_3)\]

In \( h(T_2, T_3) \), we have \( r_2(x_2) < h w_3(x_2) \) and \( r_3(x_2) < h w_2(x_2) \). This breaches the condition and therefore history \( h \) is not serializable; see Figure 9. As \( h \) is not serializable, there is no point in checking the serializability condition for the remaining \( h(T_2, T_1) \) and \( h(T_2, T_4) \).

\[
h(T_3, T_4) = r_2(x_1)w_2(x_1)r_3(x_2)w_3(x_2)r_2(x_3)w_2(x_3)r_3(x_3)w_3(x_3)\]

\[
h(T_5, T_4) = r_3(x_2)w_3(x_2)r_4(x_2)w_4(x_2)r_3(x_3)w_3(x_3)r_4(x_3)w_4(x_3)\]

\[
r_5(x_4)w_5(x_4)r_3(x_5)w_3(x_5)\]

\[
\text{Fig. 9. } G(h_T) \text{ is a subgraph of } G(h).
\]

V. APPLICATIONS

Dividing transactions into sets of steps to produce multi-step transactions improves system throughput allowing the interleaved transactions to gain more parallelism [1]. Examples of multi-step transactions are when users enter data using a sequence of forms. At the end of the sequence, the application performs updates corresponding to input data. Desktop applications using wizards are a simplification, for the user, of the same process. In e-commerce sites, checkout flow can be seen as a multi-step transaction. Booking e-tickets from travel agencies also involves executing multi-step transactions. Ticket booking from the web (or e-tickets for events, amusements, bus or flight tickets) is one of the widely available services in E-commerce. Customers can access the database and book a ticket at any time in any location.

The scenario of booking tickets is interesting in that the list of destinations are naturally ordered. It involves browsing the list of destinations, then checking the availability of seats, and booking one or more of them consecutively. For example, assume a passenger intends to book a ticket from location \( A \) to \( E \). Firstly, he/she has to browse the list of available destinations from \( A \). Secondly, choose the itinerary (the set of destinations may transit during the journey). Finally, book the itinerary. This scenario can be implemented as a multi-step transaction accessing ordered data, where a read step corresponds to browsing journey times from a destination, and the write step represents the booking phase of a chosen time to the next destination in the order. Assume that \( D \) represents the set of available destinations in the travel agency starting from \( A \) and ending at \( E \), and \( x_i \in D \) represents the next leg of the journey from \( i \), such that \( x_i \in D \). The ordered set of data items is depicted in Figure 10.

The widespread use of e-tickets makes the number of incoming and outgoing transactions of unknown even though, at any point in time, the number of active transactions in the web server is finite.

\[
\text{Fig. 10. Ordered set}
\]

VI. CONCLUSION

The emergence of mobile computing and the World Wide Web (WWW) has resulted in an indeterminate numbers of users expecting to execute their database transactions concurrently. In this paper, we have provided a serializability condition that can be used to verify the correctness of infinite histories that can model such large numbers of transactions in the case where the data accessed is ordered. The main advantage of the serializability condition which has been given, is that the testing for serializability only requires considering pairs of transactions. This makes testing for serializability efficient and easy to encode into the widely used temporal logics CTL and LTL. The modelling and verification process is one of specifying a scheduler by a transition system, encoding the serializability condition given here in temporal logic (either CTL and LTL) and then running a model checker to (automatically) perform the verification that the scheduler satisfies the serializability condition in all executions. This can all be done using common model checkers such as NuSMV and SPIN.

Further work will look to define a serializability condition for infinite histories of concurrent multi-step transactions accessing sets of data items with different graph properties which have other applications in the real-world.

ACKNOWLEDGMENT

We would like to thank ZPU (Zarqa Private University) and Loughborough University for their grant in making this work possible.

REFERENCES


