World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:11, 2010

Specifying a Timestamp-based Protocol For Multi-step Transactions Using LTL

Rafat Alshorman, Walter Hussak

Abstract—Most of the concurrent transactional protocols consider serializability as a correctness criterion of the transactions execution. Usually, the proof of the serializability relies on mathematical proofs for a fixed finite number of transactions. In this paper, we introduce a protocol to deal with an infinite number of transactions which are iterated infinitely often. We specify serializability of the transactions and the protocol using a specification language based on temporal logics. It is worthwhile using temporal logics such as LTL (Linear-time Temporal Logic) to specify transactions, to gain full automatic verification by using model checkers.

Keywords—Multi-step transactions, LTL specifications, Model Checking.

I. Introduction

RECENT advances in the development of portable devices and wireless communication networks have led to the emergence of mobile computing. In mobile computing environments, users have the opportunity to access information and services regardless of their location or movement behaviour. This means that a large community of concurrent users can submit their transactions to the database to be executed [15]. Examples of these applications are mobile auctions, stock trading and electronic commerce applications. In stock trading, submitting buy or sell transactions on the Internet has existed for some time. In electronic commerce applications, customers carrying portable devices (cell phones, laptops, PDAs) may purchase flight tickets from any airlines with their credit cards. These applications involve dealing with a huge numbers of transactions accessing databases whose consistency must be preserved in spite of updates. The component in the database system responsible for scheduling the operations of concurrent transactions to achieve consistency is the concurrency control system (or the scheduler). The scheduler orders operations belonging to different transactions by means of a concurrency control protocol [10].

The number of transactions, in most concurrency control protocols of traditional database systems, that are allowed to be executed concurrently is finite or bounded. However, in recent database systems, especially in mobile environments, the number of possible concurrent transactions is unbounded. Therefore, the schedules produced are infinite. In order to prove correctness of such systems, a verification technique that works with this view of infinite schedules is needed. Most verification techniques of such concurrency control protocols are based on finite state representations of system behaviours. These techniques cannot be directly applied to those systems of concurrent transactions where behaviours may refer to past steps of the ongoing computation or where the number of concurrent transactions is unbounded. In such cases, even simple transactions can generate infinite state systems [11].

One of the most famous examples that refers to the importance of specifying and verifying the correctness of the protocols that are used in environments where the number of users increase beyond known bounds is the Skype[1] services outage [4], [5]. This problem was caused by a massive restart of users' computers across the globe within a very short time as they rebooted after receiving a routine set of patches through Windows update [5]. A huge number of users came to the system in a continuous stream to access and request the services in terms of login transactions. The flood of attempted Skype logins together with a lack of Skype network resources, at that time, led to an outage in services. However, this event revealed a previously unseen software bug within the network resource allocation algorithm which prevented the self-healing function from working quickly. Regrettably, as a result of this disruption, Skype was unavailable to the majority of its users for approximately two days and prevented millions of registered users from accessing and making internet telephone calls using Skype software [4]. This demonstrates the limitation of some of the current protocols to deal with huge numbers of users transactions, and shows the need for specifying and verifying the correctness of the protocols that are used in such environments.

In general, mobile users submit transactions to servers that contain databases and participate in the mobile environment for execution. Since the architecture of a mobile computing system is distributed in nature [12], transactions are decomposed into a set of subtransactions, each of which executes in a different database participating in the mobile environment. Each database in a mobile environment contains a finite set of data items and therefore, the number of *different* transactions is also finite. However, the database system continuously reacts with other components in the environment in terms of transactions, and so the schedule will be infinite.

This paper is organized as follows. In Section II, we shall introduce some related and previous work. The motivation and the methodology of specifying infinite histories of multi-step transactions are discussed in Sections III and IV, respectively. In Section V, we define a protocol based on timestamps that

R. Alshorman is with the Department of computer science, Zarqa Private University, Zarqa, Jordan, e-mail: rafat_sh@zpu.edu.jo..

W. Hussak is with the Department of computer science, Loughborough University, Loughborough, LE11 3TU, UK, e-mail: W.Hussak@lboro.ac.uk.

Manuscript received July 20, 2010; revised August 20, 2010.

[1]Skype is a VoIP (Voice over Internet Protocols) telephony company that enables its users to make free voice calls with other Skype users and also low-cost calls to landlines and mobiles around the world. Recently, the number of Skype users exceeded 170 millions, about 10 millions of which are online at the same time [5]. For more information see [3].

aims to ensure the serializability of multi-step transactions accessing ordered set of data items. Linear-time temporal logic (LTL) syntax and semantics are given in Section VI. In Section VII, the properties of LTL structure for read and write operations and their interpretations on LTL paths are depicted. The LTL specifications of serializability condition and the protocol are given Section VIII, and the conclusions are drawn in Section IX.

II. Related work

In [1], a scenario of multi-step transactions has been introduced to access an ordered set of data items D. Each transaction accessed a subset of data items of a set of m data items D and each such subset accessed data items in the same order as they occurred in D. They proved the serializability condition, see Theorem 12, that represented the correctness criterion to decide whether an infinite history, composed of such transactions, is serializable by checking only all possible pairs of transactions. This results in temporal logic formulae of reduced size and scale to large number of transactions. This makes testing for serializability efficient and easy to encode into the widely used temporal logics CTL and LTL. In this paper, we shall define a protocol, based on timestamps, as a concurrency control criterion suitable for such transactions. We shall use LTL to encode the specifications for the protocol and the serializability condition. We shall introduce a method to prove that the histories produced by the timestamp-based protocol are serializable and then, we can use model checkers, such as NuSMV, to perform full automatic verifications.

III. Motivation

This paper focuses on specifying and verifying infinite histories of multi-step transactions accessing a finite set of data items with different properties, using temporal logic and model checker. Much work has been done in modelling mobile environments in order to determine performance. The aim here is to model mobile environments in order to determine correctness. In mobile computing environments infinite histories are produced, by the newer technologies of web and mobile transactions, in which transactions are continuously accessing the data items of the databases. The main desired property that needs to be specified and verified is serializability. We model a protocol that produces infinite histories in order to be able to use the NuSMV model checker based on the temporal logic LTL specifications to prove or disprove that the models satisfy this property. We define a protocol, based on timestamps, to apply the technique to, and verify its correctness for serializability.

IV. Methodology

In this section, we discuss some important methodological assumptions which will be used throughout the paper. The objective of this paper is to specify the correctness of transactions executing concurrently on a database in terms of serializability using specifications written in LTL. The reason for using temporal logics such as LTL, is that the method can be extended in order to verify infinite schedules as occur in mobile environments where transactions are incoming and outgoing in a continuous stream. The importance of temporal logic in computer science is clear, especially in the specification and verification of critical reactive systems. Model checkers, such as NuSMV, of many variants of temporal logic have been developed to the extent that they can deal with a huge number of states and verify real-world systems [16].

LTL is a temporal logic where the model of time is a path in which the future is determined. LTL is used for specifying general reactive and concurrent systems [13], [14]. It is worthwhile using LTL to specify multi-step transactions to gain full automatic verification by using model checkers.

We shall model the protocol which is used to ensure the serializability of concurrent transactions as finite state transition systems for which the specifications are expressed in LTL. Then, by exploring the state space of the state transition system, it is possible to check automatically if the protocol satisfies its desired specifications or not. The multi-step transactions model is characterized by a set of properties which ensure that any model for the protocol, produced by a model checker, should meet the defined model properties of multi-step transactions. The desired properties, that the system should satisfy, are expressed in LTL. Finally, a model checker will produce *true* if specification of the desired property satisfies all possible system behaviours. Otherwise, a counterexample will be produced to show the source of the error.

V. A Timestamp-based Protocol For Multi-step Transactions

Assume that we have an ordered set of data items D_i, where $D_i \subseteq D$, accessed by transaction $T_i \in T = \{T_j : j \in \mathbb{N}_1\}$ such that $D_i = \{x_l, x_{l+1}, \ldots, x_p\}$, see Section II of [1]. At any point in time, let $F(T_i)$ equal the first data item in D_i that is still to be accessed (the value of $F(T_i)$ will keep changing during the execution time). For any transaction T_i and data item x_a, we shall denote by $TS(T_i, x_a)$ the timestamp when T_i accesses the data item x_a (for the read operation). We assume that every timestamp value is *unique* and accurately represents an instant in time. No two timestamps can be the same. A higher-valued timestamp occurs later in time than a lower-valued timestamp. Initially, for all $x_a \in D_i$ and $T_i \in T$, $TS(T_i, x_a) = 0$. Thereafter, when T_i accesses the data item x_a, $TS(T_i, x_a)$=*System TimeStamp*. Then, the value of $TS(T_i, x_a)$ remains unchanged until the last operation in the transaction T_i ($w_i(x_p)$) has executed. Finally, when $w_i(x_p)$ has executed, the value of $TS(T_i, x_a)$ is reset to zero. Formally, we define $TS(T_i, x_a)$ as follows:

$$TS(T_i, x_a) = \begin{cases} 0, & x_a \notin D_i; \\ 0, & \text{when } T_i \text{ has executed } w_i(x_p); \\ \text{STS} & \text{when } T_i \text{ accesses } x_a; \\ TS(T_i, x_a), & \text{if } T_i \text{ has not executed } w_i(x_p) \text{ yet;} \end{cases}$$

where STS denotes the System TimeStamp. In order to execute read and write operations of transaction T_i on data item x_a, we shall compare $TS(T_i, x_{a-1})$ with the remaining transactions (on the same data item x_{a-1}). If $TS(T_i, x_{a-1})$ has the

minimum positive time stamp among all transactions that have executed x_{a-1} and are waiting to execute x_a, then T_i can access x_a. Otherwise, the transaction should suspend until it satisfies the condition (as we will see in the next subsection).

A. Accessing rules

1) There is no transaction has read x_a and has not written to x_a iff, for all $T_j \in T$,
   (a) $TS(T_j, x_a) = 0$ or
   (b) $TS(T_j, x_a) \neq 0$ and $w_j(x_a)$ has been executed.
2) T_i may access the first data item x_l in D_i, if x_l satisfies rule 1.
3) T_i may access a data item x_l, $2 \leq l \leq p$, if $F(T_i) = x_l$. Otherwise, the transaction T_i will be suspended.
4) $F(T_i) = x_l$ iff
   (a) x_l satisfies rule 1
   (b) $TS(T_i, x_{l-1}) = min\{TS(T_j, x_{l-1}) : 1 \leq j \leq n$ and $TS(T_j, x_{l-1}) > 0$ and $TS(T_j, x_l) = 0\}$
   (c) $x_l \in D_i$

For example, assume that we have 4 ordered sets of data items:

$$\begin{aligned} D_1 &= \{x_2, x_3, x_4\} \\ D_2 &= \{x_1, x_2\} \\ D_3 &= \{x_2, x_3, x_4, x_5\} \\ D_4 &= \{x_2, x_3\} \end{aligned}$$

accessed by their corresponding transactions T_1, T_2, T_3 and T_4 respectively, so that the transactions are as follows:

$$\begin{aligned} T_1 &= r_1(x_2)w_1(x_2)r_1(x_3)w_1(x_3)r_1(x_4)w_1(x_4) \\ T_2 &= r_2(x_1)w_2(x_1)r_2(x_2)w_2(x_2) \\ T_3 &= r_3(x_2)w_3(x_2)r_3(x_3)w_3(x_3)r_3(x_4)w_3(x_4)r_3(x_5)w_3(x_5) \\ T_4 &= r_4(x_2)w_4(x_2)r_4(x_3)w_4(x_3) \end{aligned}$$

and suppose that T_1 precedes T_2 and T_2 precedes T_3 and T_3 precedes T_4 in arriving at scheduler S. Also, suppose that S makes use of the above protocol to schedule the incoming transactions. The following matrix represents the Time Stamp Matrix (TSM) for all transactions in T versus all data items in D:

$$TS(T,D) = \begin{pmatrix} TS(T_1,x_1) & TS(T_1,x_2) & ... & TS(T_1,x_m) \\ TS(T_2,x_1) & TS(T_2,x_2) & ... & TS(T_2,x_m) \\ ... & ... & & ... \\ TS(T_n,x_1) & TS(T_n,x_2) & ... & TS(T_n,x_m) \end{pmatrix}$$

The entry that lies in the i^{th} row and the j^{th} column of the matrix (TMS) is typically referred to as $TS(T_i, x_j)$, and it represents the value of the timestamp for transaction T_i when it has accessed x_j. The matrix entries keep changing during the execution time. This change depends on the transactions nature (number of consecutive data items they access) and number of active transactions in any point in time. Initially, TSM will be as follows

$$TS(T,D) = \begin{pmatrix} 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \\ \vdots & \vdots & ... & \vdots \\ 0 & 0 & ... & 0 \end{pmatrix}$$

Now, assume that the schedule (history) h and TSM at some point in time are such that

$$h = r_1(x_2)r_2(x_1)w_2(x_1), \qquad TS(T,D) = \begin{pmatrix} 0 & 1 & ... & 0 \\ 2 & 0 & ... & 0 \\ \vdots & \vdots & ... & \vdots \\ 0 & 0 & ... & 0 \end{pmatrix} \quad (1)$$

and that the transaction T_3 tries to access its first data item x_2. Then, T_3 can not access the data item x_2 because it has been read by T_1 but not written yet, as by accessing rule 1. Thus, we can know whether a data item x_a has been accessed by any transaction $T_j \in T$, and also which transactions, by applying accessing rule 1. Subsequently, assume that the history h is such that

$$h = r_1(x_2)r_2(x_1)w_2(x_1)w_1(x_2),$$

the TSM is as in (1) and transactions T_2 and T_3 try to access data item x_2. As x_2 is the first data item in the set D_3, we apply accessing rule 2. T_3 will find x_2 satisfies rule 1. So, T_3 can access x_2. Also, T_2 will find x_2 satisfies accessing rule 1 and rule 4 and can be accessed by T_2 itself. Consequently, whichever one of them (T_2 or T_3) comes to the scheduler S first, can access x_2 immediately. Now, assume that the history h and TSM, at any point in time, are such that:

$$\begin{aligned} h &= r_1(x_2)r_2(x_1)w_2(x_1)w_1(x_2)r_2(x_2)w_2(x_2)r_3(x_2)w_3(x_2) \\ &\quad r_4(x_2)w_4(x_2) \end{aligned}$$

$$TS(T,D) = \begin{pmatrix} 0 & 1 & 0 & ... & 0 \\ 0 & 0 & 0 & ... & 0 \\ 0 & 4 & 0 & ... & 0 \\ 0 & 5 & 0 & ... & 0 \\ \vdots & \vdots & \vdots & ... & \vdots \\ 0 & 0 & 0 & ... & 0 \end{pmatrix}$$

and transactions T_1, T_3 and T_4 are contending to access x_3. According to accessing rule 3, any one of them that satisfies the conditions of accessing rule 4 can access x_3 (as x_3 is not the first data item in T_1, T_3 and T_4). Hence, all of them satisfy conditions (a) and (c) of rule 4. But, only T_1 satisfies also condition (b) because it has the minimum timestamp of x_2. This means that T_1 accessed x_2 first, and it should also access x_3 first. It can be easily seen that for transaction T_2, timestamps $TS(T_2, x_1)$ and $TS(T_2, x_2)$ are reset to zero. This occurs when any transaction finishes its execution on all its data items.

VI. Linear temporal logic specifications

In this section, we present LTL as a logic that can be used to specify and verify infinite histories composed of n transactions each accessing contiguous subsets of m ordered data items, and repeating infinitely often. The aggregate of all the repetitions of the n transactions gives an infinite number of transactions $T = \{T_i : i \in \mathbb{N}_1\}$. Infinite histories are produced by executing the accessing protocol of Section V, on an unlimited number of these kinds of transactions. The reason for using LTL as a specification language in this context, is that LTL formulae are interpreted over both finite and

infinite sequence of states [17], as we will see in the next sections, which is useful for the histories that are produced from executing the accessing protocol. Also, LTL is used for specifying general reactive and concurrent systems [13], [14]. We will encode the specifications of the accessing protocol, which is timestamp-based, and the serializability condition, which is introduced in Theorem 12 of [1], into LTL.

A. Syntax of LTL

The alphabet of LTL consists of a set of propositions symbols $p_0, p_1, \ldots$, distinguished read/write step propositional symbols $r_i(x_j), w_i(x_j)$, where $i \geq 1$ and $j \geq 1$, booleans $\neg, \vee, \wedge, \top, \bot$, and temporal operators **X**, **F**, **G**, **U**. Formulae in LTL are those generated by:

$$\phi \quad ::= \quad p_i \mid r_i(x_j) \mid w_i(x_j) \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \mathbf{X}\phi \mid \mathbf{F}\phi \mid \mathbf{G}\phi \mid \phi_1 \mathbf{U} \phi_2 |$$

The symbols $\bot$ and $\top$ will also be used to denote the truth values false and true respectively and the abbreviations $\Rightarrow$ and $\Leftrightarrow$ will have their usual logical meaning.

B. Semantics of LTL

An interpretation for *LTL*, $I(s_a)$, at a given state $s_a \in S$, where S is a set of states, assigns truth values $p_i^{I(s_a)}$, $r_i(x_j)^{I(s_a)}$ and $w_i(x_j)^{I(s_a)}(\in \{\bot, \top\})$ to propositional symbol p_i , $r_i(x_j)$ and $w_i(x_j)$, respectively. A Kripke structure M, as it is defined in [2], is a triple $< S,\ R,\ I >$, where S is a set of states, $R \subseteq S \times S$ a transition relation such that, for all $s \in S$, there exists $s' \in S$ with $(s, s') \in R$. A path in M is an infinite sequence of states, $\pi = s_a, s_{a+1}, \ldots$, such that, for every $b \geq a, (s_b, s_{b+1}) \in R$. We use π^a to denote the *suffix* of π starting at s_a. As each state in a Kripke structure is required to have at least one successor, it follows that $\pi^a \neq \{\}$ for any state s_a. The *semantics* of a *LTL* formula ϕ is given by the truth relation $M, s_a \vDash \phi$ which means that ϕ holds at state s_a in the Kripke structure M. Similarly, if ϕ is a path formula, $M, \pi \vDash \phi$ means that ϕ holds along path π in the Kripke structure M. The relation $\vDash$ is defined inductively as follows:

$M, s_a \vDash p_i$ iff $p_i^{I(s_a)} = \top$
$M, s_a \vDash r_i(x_j)$ iff $r_i(x_j)^{I(s_a)} = \top$
$M, s_a \vDash w_i(x_j)$ iff $w_i(x_j)^{I(s_a)} = \top$
$M, s_a \vDash \neg\phi$ iff $M, s_a \nvDash \phi$
$M, s_a \vDash \phi_1 \vee \phi_2$ iff $M, s_a \vDash \phi_1$ or $M, s_a \vDash \phi_2$
$M, s_a \vDash \phi_1 \wedge \phi_2$ iff $M, s_a \vDash \phi_1$ and $M, s_a \vDash \phi_2$
$M, s_a \vDash \mathbf{X}\phi$ iff $M, s_{a+1} \vDash \phi$
$M, s_a \vDash \mathbf{F}\phi$ iff there exists $k \geq a$ such that $M, s_k \vDash \phi$
$M, s_a \vDash \mathbf{G}\phi$ iff for all $k \geq a$ such that $M, s_k \vDash \phi$
$M, s_a \vDash \phi_1 \mathbf{U} \phi_2$ iff there exists $c \geq a$, $M, s_c \vDash \phi_2$ and, for all $a \leq b < c$, $M, s_b \vDash \phi_1$

VII. Properties of Read and Write Propositions

Assume that we have a Kripke structure M and that the following properties, relating to $r_i(x_j)$ and $w_i(x_j)$ propositions, hold in M:

(P1) Read/write alternation
A transaction T_i cannot have read two distinct data items (in D_i) without having written to one of them, i.e. if $x_j <_D x_{j'}$, $<_D$ denotes to irreflexive totally order relation on a set D, $r_i(x_{j'})$ cannot be executed until $w_i(x_j)$ has been executed.
(P2) Write implies read
A transaction T_i can only have written to x_j if it has read x_j, i.e. if $w_i(x_j)$ executes, then $r_i(x_j)$ must have executed before.
(P3) Read/write step proposition remains true until the next operation, belonging to the same transaction, becomes true
If a read/write step has taken place, the corresponding proposition remains true until the next operation in T_i become true, i.e. $r_i(x_j)/w_i(x_j)$ is true, remains true until the next step $w_i(x_j)/r_i(x_{j'})$, where $x_j <_D x_{j'}$, becomes true.
(P4) At most one step occurs at each successive state
No two distinct steps can both be false in a state, and then both true in a next state.
(P5) A transaction T_i accesses each data item $x \in D_i$ exactly once for both read and write operations
For all $x \in D_i$, a transaction T_i can have exactly one read operation ($r_i(x)$) and exactly one write operation ($w_i(x)$) for the data item x.

The semantics of formula ϕ is now given by a truth relation $M, s_a \vDash \phi$, where M is a structure for LTL satisfying the additional properties (P1)-(P4). Given a state s_a and a path π, there corresponds a sequence of read and write step propositions that become true in $s_a, s_{a+1}, \ldots$. In this way, π yields a history of the transactions $\{T_1, T_2, \ldots\}$ produced by the protocol and starting their execution at s_a. We illustrate this correspondence between paths and histories as follows:

	$\swarrow$ $r_1(x_1)$	$\swarrow$ $r_2(x_2)$	$\swarrow$ $w_1(x_1)$	$\swarrow$ $w_2(x_2)$	$\swarrow$ $r_3(x_1)$
s_0	s_1	s_2	s_3	s_4	s_5
•	•	•	•	•	•
$begin_1$	$r_1(x_1)$	$r_1(x_1)$	$w_1(x_1)$	$w_1(x_1)$	$w_1(x_1)$
$begin_2$	$begin_2$	$r_2(x_2)$	$r_2(x_2)$	$w_2(x_2)$	$w_2(x_2)$
$begin_3$	$begin_3$	$begin_3$	$begin_3$	$begin_3$	$r_3(x_1)$
$\swarrow$ $w_3(x_1)$	$\swarrow$ $r_1(x_2)$	$\swarrow$ $w_1(x_2)$	$\swarrow$ $r_3(x_2)$	$\swarrow$ $w_3(x_2)$	$\swarrow$ $r_2(x_3)$
s_6	s_7	s_8	s_9	s_{10}	s_{11}
•	•	•	•	•	•
$w_1(x_1)$	$r_1(x_2)$	$w_1(x_2)$	$w_1(x_2)$	$w_1(x_2)$	$w_1(x_2)$
$w_2(x_2)$	$w_2(x_2)$	$w_2(x_2)$	$w_2(x_2)$	$w_2(x_2)$	$r_2(x_3)$
$w_3(x_1)$	$w_3(x_1)$	$w_3(x_1)$	$r_3(x_2)$	$w_3(x_2)$	$w_3(x_2)$
$\swarrow$ $w_2(x_3)$	$\swarrow$ $r_1(x_3)$	$\swarrow$ $w_1(x_3)$	$\swarrow$ $r_3(x_3)$	$\swarrow$ $w_3(x_3)$	$\swarrow$ $r_2(x_4)$
s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}
•	•	•	•	•	•
$w_1(x_2)$	$r_1(x_3)$	$w_1(x_3)$	$w_1(x_3)$	$w_1(x_3)$	$w_1(x_3)$
$w_2(x_3)$	$w_2(x_3)$	$w_2(x_3)$	$w_2(x_3)$	$w_2(x_3)$	$r_2(x_4)$
$w_3(x_2)$	$w_3(x_2)$	$w_3(x_2)$	$r_3(x_3)$	$w_3(x_3)$	$w_3(x_3)$

$\swarrow$	$\swarrow$	$\swarrow$	$\swarrow$	$\swarrow$	$\swarrow$
$w_2(x_4)$	end_1	end_2	$r_3(x_4)$	$w_3(x_4)$	$r_3(x_5)$
s_{18}	s_{19}	s_{20}	s_{21}	s_{22}	s_{23}
$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
$w_1(x_3)$	end_1	end_1	end_1	end_1	end_1
$w_2(x_4)$	$w_2(x_4)$	end_2	end_2	end_2	end_2
$w_3(x_3)$	$w_3(x_3)$	$w_3(x_3)$	$r_3(x_4)$	$w_3(x_4)$	$r_3(x_5)$

$\swarrow$	$\swarrow$
$w_3(x_5)$	end_3
s_{24}	s_{25}
$\bullet$	$\bullet$
end_1	end_1
end_2	end_2
$w_3(x_5)$	end_3

In the depiction above, we have

$$\begin{aligned} D_1 &= \{x_1, x_2, x_3\} \\ D_2 &= \{x_2, x_3, x_4\} \\ D_3 &= \{x_1, x_2, x_3, x_4, x_5\} \end{aligned}$$

and the corresponding transactions as follows

$$\begin{aligned} T_1 &= r_1(x_1)w_1(x_1)r_1(x_2)w_1(x_2)r_1(x_3)w_1(x_3) \\ T_2 &= r_2(x_2)w_2(x_2)r_2(x_3)w_2(x_3)r_2(x_4)w_2(x_4) \\ T_3 &= r_3(x_1)w_3(x_1)r_3(x_2)w_3(x_2)r_3(x_3)w_3(x_3) \\ &\quad\ r_3(x_4)w_3(x_4)r_3(x_5)w_3(x_5) \end{aligned}$$

The read and write propositions that are given for each successive state represent the propositions that are true in those states. The top of each column displays the unique proposition that becomes true in the particular state. This represents the read and write operations that has been scheduled by the protocol. In order to make it easier to follow the structure which has a large number of propositions which are false, only the values of propositions that are true in the states of the trace are shown. The corresponding history h is:

$$\begin{aligned} h = {} & r_1(x_1)r_2(x_2)w_1(x_1)w_2(x_2)r_3(x_1)w_3(x_1)r_1(x_2)w_1(x_2) \\ & r_3(x_2)w_3(x_2)r_2(x_3)w_2(x_3)r_1(x_3)w_1(x_3)r_3(x_3)w_3(x_3) \\ & r_2(x_4)w_2(x_4)r_3(x_4)w_3(x_4)r_3(x_5)w_3(x_5) \quad . \end{aligned}$$

We make use of additional propositions $begin_i$ and end_i to refer to the begin and the end of each transaction.

VIII. Encoding the accessing protocol and serializability condition into LTL

Firstly, we encode, using temporal operators, the properties (P1)-(P5) of the read and write propositions in the LTL structures (as in the previous section) as σ_0, σ_1, σ_2, σ_3 and σ_4 respectively, as follows

(P1) Read/write alternation

A transaction T_i cannot have read two distinct data items (in D_i) without having written to one of them, i.e. if $x <_D y$, $r_i(y)$ cannot be executed until $w_i(x)$ has been executed.

$$\sigma_0 = \bigwedge_{i\geq 1} \bigwedge_{x,y\in D_i, x<_D y} \mathbf{G}[(r_i(x) \Rightarrow \mathbf{F}(w_i(x) \wedge \mathbf{F}(r_i(y))))]$$

(P2) Write implies read

A transaction T_i can only have written to x if it has read x, i.e. if $w_i(x)$ executes, then $r_i(x)$ must have executed before.

$$\sigma_1 = \bigwedge_{i\geq 1} \bigwedge_{x\in D_i} \mathbf{G}[r_i(x) \Rightarrow \mathbf{F}(w_i(x))]$$

(P3) Read/write step proposition remains true until the next operation, belonging to the same transaction, becomes true

If a read/write step has taken place, the corresponding proposition remains true until the next operation in T_i becomes true, i.e. if $r_i(x)/w_i(x)$ is true, it remains true until the next step $w_i(x)/r_i(y)$, where $x <_D y$, becomes true.

$$\begin{aligned} \sigma_2 = {} & \bigwedge_{i\geq 1} \bigwedge_{x\in D_i} \mathbf{G}[w_i(x) \Rightarrow \neg r_i(x)] \quad \wedge \\ & \bigwedge_{i\geq 1} \bigwedge_{x,y\in D_i, x<_D y} \mathbf{G}[r_i(y) \Rightarrow \neg w_i(x)] \end{aligned}$$

(P4) At most one step occurs at each successive state

No two distinct steps can both be false in a state, and then both true in a next state.

$$\begin{aligned} \sigma_3 = {} & \bigwedge_{\substack{i,i'\geq 1 \\ 1\leq j,j'\leq m \\ i\neq i' \text{ or } j\neq j'}} \\ & \mathbf{G}\ [\neg((\neg r_i(x_j) \wedge \neg r_{i'}(x_{j'})) \wedge \mathbf{X}(r_i(x_j) \wedge r_{i'}(x_{j'}))) \\ & \wedge\neg((\neg r_i(x_j) \wedge \neg w_{i'}(x_{j'})) \wedge \mathbf{X}(r_i(x_i) \wedge w_{i'}(x_{j'}))) \\ & \wedge\neg((\neg w_i(x_j) \wedge \neg w_{i'}(x_{j'})) \wedge \mathbf{X}(w_i(x_j) \wedge w_{i'}(x_{j'})))]. \end{aligned}$$

(P5) A transaction T_i accesses each data item $x \in D_i$ exactly once for both read and write operations

For all $x \in D_i$, a transaction T_i can have exactly one read operation ($r_i(x)$) and exactly one write operation ($w_i(x)$) for the data item x, i.e we can not have T_i such that

$$T_i = r_i(x_1)w_i(x_1)\ldots r_i(x_1)\ldots w_i(x_p)$$

or

$$T_i = r_i(x_1)w_i(x_1)\ldots w_i(x_1)\ldots w_i(x_p)$$

where $D_i = \{x_1, x_2, \ldots, x_p\}$. This is defined in σ_5 as follows

$$\begin{aligned} \sigma_4 = {} & (\bigwedge_{i\geq 1} \bigwedge_{x\in D_i} \mathbf{G}\neg[r_i(x) \wedge \mathbf{F}(\neg r_i(x) \wedge \mathbf{F}r_i(x))]) \wedge \\ & (\bigwedge_{i\geq 1} \bigwedge_{x\in D_i} \mathbf{G}\neg[w_i(x) \wedge \mathbf{F}(\neg w_i(x) \wedge \mathbf{F}w_i(x))]) \end{aligned}$$

Next, we encode the serializability condition of Theorem 12 (see [1]). This is defined in terms of σ_5 and σ_6:

$$\begin{aligned} \sigma_5 = {} & \bigwedge_{i,i'\geq 1, i\neq i'} \bigwedge_{x,y\in D_i\cap D_{i'}, x<_D y} \\ & \mathbf{G}\neg[(w_i(x) \wedge \mathbf{F}(w_{i'}(x) \wedge \mathbf{F}(w_{i'}(y) \wedge \mathbf{F}(w_i(y))))) \vee \\ & (w_{i'}(x) \wedge \mathbf{F}(w_i(x) \wedge \mathbf{F}(w_i(y) \wedge \mathbf{F}(w_{i'}(y))))))] \end{aligned}$$

$$\sigma_6 = \bigwedge_{i,i'\geq 1, i\neq i'} \bigwedge_{x\in D_i\cap D_{i'}} \mathbf{G}\neg[r_i(x) \wedge r_{i'}(x)]$$

The serializability condition of Theorem 12 says if we choose any two transactions (T_i and $T_{i'}$) participating in a history h and we take the history that containing only the operations that belonging to T_i and $T_{i'}$ (denoted by $h_{\{T_i,T_{i'}\}}$), where $D_i \cap D_{i'} = \{x_l, \ldots, x_p\}$. Then, $h_{\{T_i,T_{i'}\}}$ will be of the form

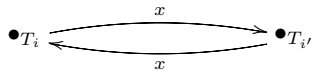
$$\begin{aligned} h_{\{T_i,T_{i'}\}} = \quad & \ldots r_i(x_l) \ldots w_i(x_l) \ldots r_{i'}(x_l) \ldots w_{i'}(x_l) \ldots \\ & r_i(x_p) \ldots w_i(x_p) \ldots r_{i'}(x_p) \ldots w_{i'}(x_p) \ldots \end{aligned}$$

then the the history h is serializable. In σ_6, we encode that if a transaction T_i begins executing a read operation on data item x, no read operation on data item x by any other transactions occurs (executes) until the write operation of T_i completes its execution on data item x, i.e.:

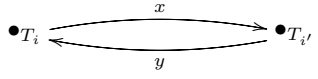
$$\ldots r_i(x) \underbrace{\ldots\ldots\ldots\ldots\ldots}_{\text{no } r_{i'}(x)} w_i(x) \ldots$$

Therefore, if we avoid the situation above, there is no cycle in the precedence graph G of a history $h_{\{T_i,T_{i'}\}}$ (denoted by $G(h_{\{T_i,T_{i'}\}})$ as in [1], [2], [10]) between T_i and any other transaction $T_{i'}$ on the *same* data item; see Figure 1(a). Now, the serializability condition, of Theorem 12, is to hold for each $x \in D_i \cap D_{i'}$. It is possible to make a cycle of length two in a precedence graph $G(h_{\{T_i,T_{i'}\}})$ on different data items, i.e. if T_i precedes $T_{i'}$ in accessing data item x and $T_{i'}$ precedes T_i in accessing data item y, or $T_{i'}$ precedes T_i in accessing data item x and T_i precedes $T_{i'}$ in accessing data item y, see Figure 1(b) and Figure 1(c). Therefore, we encode in σ_5 that these two situations do not occur. The formulae σ_5 and σ_6 together represent the encoding of the serializability condition into LTL denoted by σ_7:

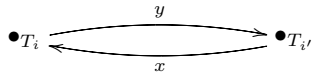
$$\sigma_7 = \sigma_5 \wedge \sigma_6$$



(a) Cycle on the same data item



(b) Cycle on different data items



(c) Cycle on different data items

Fig. 1. Cycle of length two

Next, we encode the accessing protocol, which defines in section V, in σ_{10} as follows:

$$\begin{aligned} \sigma_8 = & \bigwedge_{\substack{i,j,k \geq 1 \\ i \neq j, j \neq k, i \neq k}} \\ & \mathbf{G}[(\bigwedge_{x \in D_i \cap D_j \cap D_k} (w_i(x) \wedge \mathbf{F}(w_j(x) \wedge \mathbf{F}(w_k(x))))) \vee \\ & (\bigwedge_{x \in D_i \cap D_j \cap D_k} (w_i(x) \wedge \mathbf{F}(w_k(x) \wedge \mathbf{F}(w_j(x))))) \vee \\ & (\bigwedge_{x \in D_i \cap D_j \cap D_k} (w_k(x) \wedge \mathbf{F}(w_i(x) \wedge \mathbf{F}(w_j(x)))))]. \end{aligned}$$

In σ_8, we model an unbounded number of transactions, that may come to the scheduler S, which uses the protocol that is defined in section V, by three transactions T_i, T_j and T_k. Transactions T_i and T_j represent any two particular transactions satisfying the accessing rules of the protocol and T_k represents any other transaction in the schedule S. Transaction T_k could execute any data item $x \in D_i \cap D_j \cap D_k$ as follows

- Case 1: T_k could execute x after T_i and T_j

$$\ldots \quad \underset{w_i(x)}{\bullet} \quad \ldots \quad \underset{w_j(x)}{\bullet} \quad \ldots \quad \underset{w_k(x)}{\bullet} \quad \ldots$$

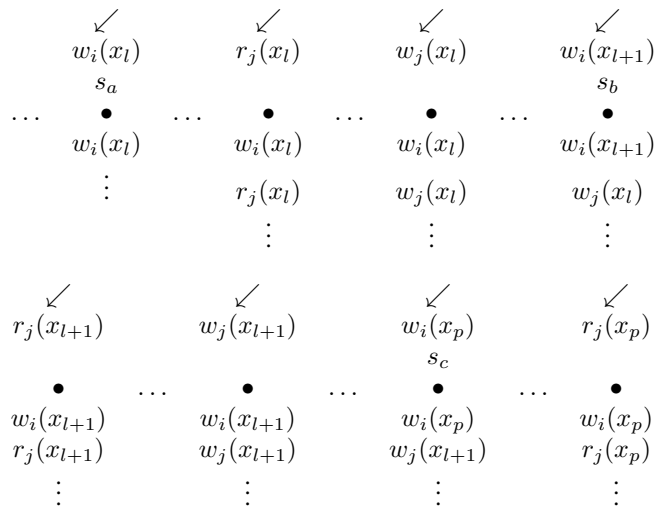
- Case 2: T_k could execute x after T_i and before T_j

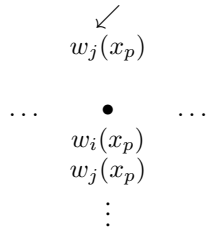
$$\ldots \quad \underset{w_i(x)}{\bullet} \quad \ldots \quad \underset{w_k(x)}{\bullet} \quad \ldots \quad \underset{w_j(x)}{\bullet} \quad \ldots$$

- Case 3: T_k could execute x before T_i and T_j

$$\ldots \quad \underset{w_k(x)}{\bullet} \quad \ldots \quad \underset{w_i(x)}{\bullet} \quad \ldots \quad \underset{w_j(x)}{\bullet} \quad \ldots$$

In the depiction above, we assume that T_i executes the data item x, where $x \in D_i \cap D_j$, before T_j does. Therefore, we can say that, T_i and T_j satisfy the accessing rules of the protocol if and only if for all $x \in D_i \cap D_j$, $TS(T_i, x) < TS(T_j, x)$ or $TS(T_j, x) < TS(T_i, x)$ regardless of when the transaction T_k could access the data item x. We can illustrate this in the LTL structure as follows:





where $D_i \cap D_j = \{x_l, x_{l+1}, \ldots, x_p\}$. We notice, from the depiction above, that transaction T_i has executed $w_i(x_l)$ and T_j has not executed both $r_j(x_l)$ and $w_j(x_l)$ yet, regardless of when/where the transaction T_k executes read and write operations on the data item x_l and similarly for x_{l+1} up to x_p. Transaction T_j will execute both $r_j(x_l)$ and $w_j(x_l)$ after T_i does and this will keep T_i and T_j serializable. In other words, execution of any operations belonging to other transactions will not affect the serializability of the transactions T_i and T_j.

The access rules of the protocol (see subsection V.A) say that if a transaction T_i has executed $r_i(x)$ but not $w_i(x)$, no other transaction T_j can execute $r_j(x)$. This can be encoded into LTL as follows

$$\sigma_9 = \bigwedge_{i,i' \geq 1, i \neq i'} \bigwedge_{x \in D_i \cap D_{i'}} \mathbf{G}[r_i(x) \Longrightarrow \neg r_{i'}(x)]$$

Therefore σ_{10}, which represents the access protocol, is defined as follows

$$\sigma_{10} = \sigma_8 \wedge \sigma_9.$$

This means that any history h produced by the access protocol should satisfy both σ_8 and σ_9, if we model the read and write operations in the history h by an LTL structure.

Now, the future in LTL is seen as a sequence of states, so the future is a path. Therefore, if we consider the states in the LTL path as instants of time, we can assign a truth value to each proposition at each time instant so that the interpretation maps to each instant of time a set of propositions that hold at that instant. As a consequence of this, we can assign a truth value to each read/write proposition belonging to any active transaction, at any given time, that is scheduled by the protocol. As we are dealing with a protocol based on timestamps, our interpretation will, therefore, map each timestamp to a set of propositions that hold at that timestamp. Thus, the interpretation I is a function

$$I : \mathbb{N} \rightarrow 2^{prop}$$

where $\mathbb{N}$ is a set of timestamps (natural number) and $prop$ is a set of all propositions. This means that we can specify infinite histories generated by a protocol based on timestamps using LTL specifications. Now, as σ_{10} represents the histories produced (or generated) by the protocol and σ_7 represents the serializability condition then, we can prove that the histories are serializable by proving the following formula:

$$\sigma_{10} \Longrightarrow \sigma_7.$$

IX. Conclusion

We have given an approach, using LTL, to specify and verify a scheduler uses a protocol, based on timestamp, to schedule an unlimited number of transactions incoming and outgoing from a system. This approach allows for the systems and their properties to be specified using temporal logics. The verification part can be performed by model checkers, such as NuSMV or Spin, to check exhaustively the state space of the system behaviours against the specifications. If the system does not satisfy its properties, counterexamples to pinpoint the errors are automatically produced by the model checker.

we have assumed serializability to be the correctness criterion for concurrent transactions executing in a transactions processing system. Also, we have shown that the serializability condition and the scheduler can be encoded into LTL. Then, an automatic verification can be performed using a model checker, to see whether the scheduler satisfies the serializability condition or not.

We have found that LTL is a good choice to specify a protocol based on timestamps because any state in LTL path could implicitly represent a timestamp. Actually, we encoded the behaviour of the protocol in terms of three transactions. Two of them represent any two transactions satisfying the accessing rules of the protocol, and the remaining one represents any other transaction in the schedule. This means that we can prove that the schedules, produced by the protocol, are serializable if

$$\sigma_{10} \Longrightarrow \sigma_7$$

is satisfied, where σ_{10} represents the LTL formula that specifies the behaviour of the protocol in LTL structure, that we have given above, and σ_7 represents the LTL formula that specifies the serializability condition. This gives us an automatic verification approach that can overcome the disadvantages of the traditional approaches such as human error and may not cover all possible system behaviours.

Acknowledgment

We would like to thank ZPU (Zarqa Private University) for its support in making this work possible.

References

[1] R. Alshorman and W. Hussak, *A Serializability Condition for Multi-step Transactions Accessing Ordered Data*, International Journal of Computer Science, Vol. 4, issue 1 (2009), pp. 13-20.
[2] R. Alshorman and W. Hussak, *A CTL Specification of Serializability for Transactions Accessing Uniform Data*, International Journal of Computer Science and Engineering , Vol. 3, issue 1 (2009), pp. 26-32.
[3] *Skype Web site*, http://www.skype.com
[4] *Skype Heartbeats Archives*, http://heartbeat.skype.com/2007/08/
[5] D. Rossi, M. Mellia and M. Meo, *Evidences Behind Skype Outage*, In proceedings of the IEEE International Conference on Communication (ICC'09), Dresde, Germany, June 2009, Link: http://www.tlc-networks.polito.it/mellia/papers/Skype_outage_icc09.pdf
[6] A. Cimatti, E. Clarke, F. Giunchiglia and M. Roveri, *NuSMV: a new symbolic model verifier*, In proceedings of the 11th International Conference on Computer Aided Verification, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1633, 1999, pp. 495-499.
[7] *NuSMV v2.4 Tutorial*, http://nusmv.fbk.eu/NuSMV/tutorial/v24/tutorial.pdf, NuSMV website.
[8] A. Cimatti, E. Clarke, F. Giunchiglia and M. Roveri, *NuSMV: a new symbolic model verifier*, In proceedings of the 11th International Conference on Computer Aided Verification, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1633, 1999, pp. 495-499.
[9] *NuSMV v2.4 Tutorial*, http://nusmv.fbk.eu/NuSMV/tutorial/v24/tutorial.pdf, NuSMV website.
[10] R. Elmasri, S. Navathe, *Fundamental of Database Systems*, Addison-Wesley, Fourth Edition, 2004.

[11] S. Gnesi, *Formal Specification and Verification of Complex Systems*, Electronic Notes in Theoretical Computer Science Netherlands, Vol. 80, 2003, pp. 294-298.
[12] W-C. Peng and M-S. Chen, *Mining user moving patterns for personal data allocation in a mobile computing system*, In IEEE proceedings of 29th International Conference on Parallel Processing, 2000, pp. 573580.
[13] Z. Manna and A. Pnueli, *Temporal verification of reactive systems: Safety*, Springer-Verlag N.Y. Inc., 1995.
[14] K. Sen, G. Rosu and G. Agha, *Generating Optimal Linear Temporal Logic Monitors by Coinduction*, In proceedings of 8th Asian Computing Science Conference (ASIAN03), Lecture Notes in Computer Science, Springer-Verlag, Vol. 2896, 2003, pp. 260-275.
[15] V.C.S. Lee, K-W. Lam, S.H. Son and E.Y.M. Chan, *On transaction processing with partial validation and timestamp ordering in mobile broadcast environments*, IEEE Transactions on Computers, Vol. 51, issue 10 (2002), pp. 1196-1211.
[16] R. Alshorman and W. Hussak, *Multi-step transactions specification and verification in a mobile database community*, In proceedings of 3rd IEEE International Conference on Information Technologies: from Theory to Applications, IEEE, ICTTA 08, Damacus, Syria, IEEE Computer Society Press, 2008, pp. 1407-12.
[17] R. Pucella, *The finite and the infinite in temporal logic*, ACM SIGACT News, Vol. 36, issue 1 (2005), pp. 86-99.