A Formal Specification of the Concurrency Control in Real-Time Databases

Ekaterina Pavlova[*]
St.-Petersburg State University, Russia
katya@meta.math.spbu.ru

Dang Van Hung
UNU/IIST, P.O. Box 3058, Macau
dvh@iist.unu.edu

Abstract

In the paper we present a formal model of real-time database (RTDB) systems using Duration Calculus (DC). First, we give a formal specification of the correctness for the executions of transaction systems and the Two Phase Locking Concurrency Control Protocol (2PL-CCP). We also give a formal proof for the correctness of the 2PL-CCP using the DC proof system. Then, we present a formal description of the real-time database model by extending the model for untimed databases with state variables expressing temporal objects and with DC formulas to express their behaviour. A formal description of correctness of the parallel executions of transaction systems in RTDBs is then given as the combination of the correctness for the untimed case and the time constraints for the transactions and their read data.

1 Introduction

As computers have become faster and more powerful, and as their use has become more widespread, we have to handle large database systems (DBSs) in which time is critical. In this kind of databases the transactions not only have to meet their deadline, but also have to use the data that are valid during their execution. This caused the appearance of Real-Time Database Systems (RTDBSs) [1, 8] in which we have to deal with some real-time requirements in addition to the traditional ones. Therefore, the design of real-time databases, in particularly the concurrency control protocols in the systems, is rather complicated, and hence the use of formal methods in the design of real-time databases should be strongly recommended to reduce number of faults and to increase the reliability. To our knowledge, formal methods are widely used in other areas but it is still not the case in the area of Real-Time Database Systems.

Our purpose in this paper is to formalise some aspects of RTDBs, in particular the Concurrency Control, using a real-time logic. This will allow us to verify the correctness of concurrency control protocols [9, 10] formally using the proof system of the logic. Because Duration Calculus (DC), introduced by Zhou, Hoare and Ravn [2] in 1991, is a simple and powerful logic for reasoning about real-time systems, and because DC has been used successfully in many case studies, for example [4], we will take it to be the formalism for our specification in this paper.

Our approach is summarised as follows. At first we use boolean functions of time, which are called state variables in DC, to model the timed behaviour of data items and operations of transactions. A specification of the transaction and data management systems is then written as DC formulas on these state variables. As a result, the correctness for the concurrency control in untimed databases is expressed precisely as a DC formula. To show the advantage of our model, we give a formal specification of the well-known Two Phase Locking Concurrency Control Protocol [9, 10] (2PL-CCP) in DC and a formal proof of its correctness using the DC proof system. Then, we extend the model to handle the time issues by adding some state variables and DC formulas to express the temporal validity of the data and the real-time requirements for the transactions. A formal description of correctness criterion of the parallel executions of transaction systems in RTDBs is then just the conjunction of the correctness criterion for the untimed case and the time constraints for the transactions and their read data.

The paper is organised as follows. In the next section, we give an informal abstract description of DBs and RTDBs. A brief summary of DC is given in Section 3. Section 4 presents our formal model for RTDBs and the concurrency control problem. The last section is the conclusion of the paper.

2 Preliminaries

We briefly recall in this section the main concepts of DBS, RTDBS and the concurrency control problems, which will justify our formal model given in later sections. We start with the concept of DBS, and then describe features of

[*]The paper was prepared during the fellowship of the first author at The United Nations University, International Institute for Software Technology (UNU/IIST) from September 1, 1998 to March 1, 1999

RTDBS and the main difference between conventional DBS and RTDBS [10, 1].

A database is an entity that integrates and stores related data in an organised manner. Actually, the data stored in the DB represent the state of DB at a single moment of time. Because the contents of the DB change when information is added and out-of-date data is deleted from the DB, we can consider the DB as set of variables which are changing in time.

The access of users to the DB is by executing a transaction which is a logical sequence of read and write operations. A transaction that updates the data objects of the DB must preserve the integrity constraints of the DB (these integrity constraints are often expressed as logical formulas on the data objects (variables)). Note that a transaction may violate some integrity constraints during its execution. However, once it terminates, it must restore the DB to a consistent state.

Traditionally, transactions are expected to satisfy the execution atomicity meaning that the execution of any transaction is free from interference with the others.

A protocol that ensures the execution atomicity is called *concurrency control* protocol (CCP). A CCP is considered to be correct iff any execution of the transactions controlled by it satisfies the correctness criterion. From the consistency of the transactions it is obvious that a serial execution of transactions will take the DB from a consistent state to a consistent one, and hence it is correct. The standard notion of correctness for the (concurrent) executions of transactions in DBS is the *serializability* (see, e.g., [10]) which requires the effects of transactions must be same as that produced by executing the set of transactions in some serial order.

In most CCPs, the serializability is based on the notion of conflicting operations and is called *conflict serializability*. Two operations are said to be in conflict iff they are on the same data item (variable), and at least one of them is a write. Then, a concurrent execution of transactions is serializable iff there is a total order on the set of transactions such that if a transaction T is before a transaction T' according to this order, then any operation O of T should be before those operations O' of T' that are in conflict with O according to the concurrent execution. There are several other definitions of the serializability, e.g. *view serializability*, *order-preserving conflict serializability*, *one-version serializability*. There are also different CCPs for each known serializability. In this paper we consider only the *conflict serializability* and the Two Phase Locking CCP (2PL-CCP) [10] which are most commonly used in the DBSs.

According to 2PL-CCP, each data object has two kinds of locks on it: *read* locks and *write* locks. A transaction can read (write) on a data object x if only if it has a read lock (a write lock) on x. Two locks are said to be in conflict iff they are on the same data object and one of them is a write lock. Transactions can only share the non-conflict locks. In 2PL-CCP an execution of a transaction consists of two phases. In the first phase (called obtaining phase) the locks on the data objects needed by the transaction are acquired and no lock is released, while in the second phase locks are released and new locks cannot be acquired.

The executions of transactions according to 2PL-CCP are serializable. However, to our knowledge, it seems that no formal proof is given to this fact.

A Real-Time Database system (RTDBS) can be viewed as an amalgamation of a conventional Database Management System (DBMS) and a real-time system. In addition to the need of serializability, the transactions are required to operate in real-time, satisfying timing constraints imposed on transaction commitments (deadlines) and on the temporal validity of data (validity intervals) [12, 11].

In such RTDBs the set of data objects consists of non-temporal and temporal ones. A temporal data objects reflect the status of objects in the real world. Each value of a temporal data object may become invalid with the passage of time. Therefore, there are two different representations of such data objects: an *external* representation (in the real world) and *internal* representation (in the DB). Both representations are temporally related with each other which are said to be temporal consistent. The temporal consistency consists of two parts: absolute and relative. The absolute temporal consistency represents the requirement of data freshness. This means that the actual state of the external world (external representation) and the state represented by the contents of the database (internal representation) must be close enough to remain within the tolerance limit of applications. The relative temporal consistency represents the required correlation among data that are used together. This means that data read by transactions must have approximately the same age. A graphical representation of absolute and relative temporal consistency of data is shown in figure 1.

In RTDBS the transaction processing is complicated because it requires an integrated set of protocols that must guarantee not only to maintain the consistency of the database but also to operate under timing constraints. In particular, RTDBSs require new CCPs which take into consideration the deadlines and other timing constraints associated with transactions, and also the fact that data read by a transaction must be valid at the time the transaction reads them and must stay valid until completion of the transaction. These requirements make existing CCP for conventional DB unacceptable for RTDBS. Most of CCPs for RTDBS are extensions of protocols for conventional DB [5, 6, 7], but no of them can guarantee the satisfaction of all timing requirements.

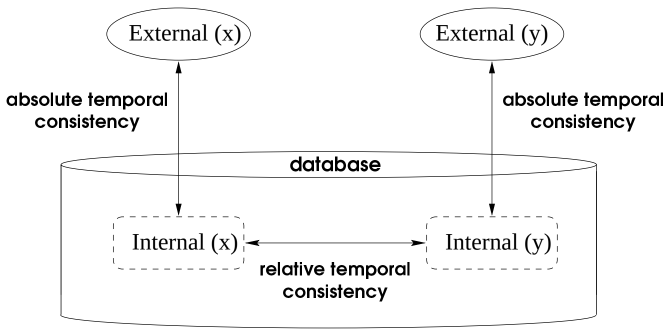
External (x)
External (y)
absolute temporal consistency
absolute temporal consistency
database
Internal (x)
Internal (y)
relative temporal consistency

Figure 1. Absolute and relative temporal consistency of temporal data

3 A Brief Summary of Duration Calculus

In this section, we give a brief summary of Duration Calculus (DC) which will be used as the formalism to specify the models of DBS and RTDBS in this paper. For more details, readers are referred to [2].

Time in DC is the set R^+ of non-negative real numbers. For $t, t' \in R^+, t \leq t', [t, t']$ denotes the time interval from t to t'.

We assume a set E of boolean state variables. E includes the Boolean constants 0 and 1 denoting *false* and *true* respectively. State expressions, denoted by P, Q, P_1, Q_1, etc., are formed by the following rules:

1. Each state variable $P \in E$ is a state expression.
2. If P and Q are state expressions, then so are $\neg P$, $(P \wedge Q), (P \vee Q), (P \Rightarrow Q), (P \Leftrightarrow Q)$.

A state variable P is interpreted as a function $I(P) : R^+ \rightarrow \{0, 1\}$ (a state). $I(P)(t) = 1$ means that state P is present at time instant t, and $I(P)(t) = 0$ means that state P is not present at time instant t. We assume that a state has finite variability in a finite time interval. A state expression is interpreted as a function which is defined by the interpretations for the state variables and Boolean operators.

For an arbitrary state expression P, its duration is denoted by $\int P$. Given an interpretation I of state variables and an interval, duration $\int P$ is interpreted as the accumulated length of time within the interval at which P is present. So for an arbitrary interval $[t, t']$, the interpretation $I(\int P)([t, t'])$ is defined as $\int_t^{t'} I(P)(t)dt$. Therefore, $\int 1$ always gives the length of the intervals and is denoted by ℓ.

The set of primitive duration terms consists of variables over the set R^+ of non-negative real numbers and durations of state variables.

A primitive duration formula is an expression formed from terms by using the usual relational operations on the reals, such as equality = and inequality $<$. A duration formula is either a primitive formula or an expression formed from formulas by using the logical operators $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, and the chop $\frown$ (see below) and quantifiers $\forall, \exists$ applied to variables ranging over R^+.

A duration formula D is satisfied by an interpretation I in an interval $[t', t'']$ just when it evaluates to true for that interpretation over that time interval. This is written as

$$I, [t', t''] \models D \ ,$$

where I assigns every state variable a finitely variable function from R^+ to $\{0,1\}$.

Given an interpretation I, the chop-formula $D_1 \frown D_2$ is true for $[t', t'']$ if there exists a t such that $t' \leq t \leq t''$ and D_1 and D_2 are true for $[t', t]$ and $[t, t'']$ respectively.

We give now shorthands for some duration formulas which are often used. For an arbitrary state variable P, $\lceil P \rceil$ stands for $(\int P = \ell) \wedge (\ell > 0)$. This means that interval is a non-point interval and P holds almost everywhere in it. We use $\lceil\, \rceil$ to denote the predicate which is true only for point intervals and define $\lceil P \rceil^* \mathrel{\widehat{=}} \lceil\, \rceil \vee \lceil P \rceil$. Modalities $\Diamond, \Box$ are defined as: $\Diamond D \mathrel{\widehat{=}} true \frown D \frown true$, $\Box D \mathrel{\widehat{=}} \neg \Diamond \neg D$. This means that $\Diamond D$ is true for an interval iff D holds for some subinterval of it, and $\Box D$ is true for an interval iff D holds for all subintervals of it.

DC with abstract duration domain is a complete calculus, which has a powerful proof system. Here we give only some rules and axioms that will be used later in this paper.

$$\begin{array}{l} \text{if } A \Rightarrow B \text{ then } (A \frown C \;\Rightarrow\; B \frown C) \\ \text{if } A \Rightarrow B \text{ then } (C \frown A \;\Rightarrow\; C \frown B) \end{array} \qquad \text{(DC-1)}$$

$$(A \frown B) \frown C \;\Leftrightarrow\; A \frown (B \frown C) \qquad \text{(DC-2)}$$

$$(A \frown \lceil\, \rceil) \;\Leftrightarrow\; (\lceil\, \rceil \frown A) \;\Leftrightarrow\; A \qquad \text{(DC-3)}$$

$$(A \frown \mathit{false}) \;\Leftrightarrow\; (\mathit{false} \frown A) \;\Leftrightarrow\; \mathit{false} \qquad \text{(DC-4)}$$

$$\begin{array}{lcl} (A \vee B) \frown C & \Leftrightarrow & (A \frown C) \vee (B \frown C) \\ C \frown (A \vee B) & \Leftrightarrow & (C \frown A) \vee (C \frown B) \end{array} \qquad \text{(DC-5)}$$

$$\begin{array}{lcl} (A \wedge B) \frown C & \Rightarrow & (A \frown C) \wedge (B \frown C) \\ C \frown (A \wedge B) & \Rightarrow & (C \frown A) \wedge (C \frown B) \end{array} \qquad \text{(DC-6)}$$

$$\Box A \;\Rightarrow\; A \tag{DC-7}$$

$$\lceil P \rceil ^\frown \lceil P \rceil \Leftrightarrow \lceil P \rceil \tag{DC-8}$$

$$\lceil P \rceil \;\Rightarrow\; \Box(\lceil P \rceil^*) \tag{DC-9}$$

$$\lceil P \rceil \wedge \lceil Q \rceil \;\Leftrightarrow\; \lceil P \wedge Q \rceil \tag{DC-10}$$

$$\begin{array}{l} \lceil P \rceil ^\frown true \;\wedge\; true ^\frown \lceil \neg P \rceil \;\Rightarrow \\ \lceil P \rceil ^\frown true ^\frown \lceil \neg P \rceil \end{array} \tag{DC-11}$$

$$\begin{array}{l} \lceil P \rceil ^\frown true ^\frown \lceil \neg P \rceil \;\Rightarrow \\ \lceil P \rceil ^\frown \lceil \neg P \rceil ^\frown true \wedge true ^\frown \lceil \neg P \rceil ^\frown \lceil P \rceil \end{array} \tag{DC-12}$$

$$\begin{array}{l} \lceil P \rceil ^\frown true \;\wedge\; true ^\frown \lceil \neg P \rceil ^\frown A \;\Rightarrow \\ \lceil P \rceil ^\frown true ^\frown \lceil \neg P \rceil ^\frown A \end{array} \tag{DC-13}$$

$$\begin{array}{l} A ^\frown \lceil P \rceil ^\frown true \;\wedge\; true ^\frown \lceil \neg P \rceil \;\Rightarrow \\ A ^\frown \lceil P \rceil ^\frown true ^\frown \lceil \neg P \rceil \end{array} \tag{DC-14}$$

$$\begin{array}{l} \lceil P \rceil \;\wedge\; \lceil Q_1 \rceil ^\frown \lceil Q_2 \rceil \;\Leftrightarrow \\ (\lceil P \rceil \wedge \lceil Q_1 \rceil) ^\frown (\lceil P \rceil \wedge \lceil Q_2 \rceil) \end{array} \tag{DC-15}$$

$$\begin{array}{l} \lceil P \rceil ^\frown (\lceil \neg P \rceil \wedge A) \wedge \lceil P \rceil ^\frown (\lceil \neg P \rceil \wedge B) \\ \Rightarrow\; \lceil P \rceil ^\frown (\lceil \neg P \rceil \wedge A \wedge B) \end{array} \tag{DC-16}$$

$$\neg \lceil P \rceil \;\Leftrightarrow\; \lceil\,\rceil \vee \Diamond \lceil \neg P \rceil \tag{DC-17}$$

$$\Box(A) ^\frown \Box(B) \;\Rightarrow\; \Box(A \vee B \vee A ^\frown B) \tag{DC-18}$$

4 A Formalisation of Real-Time Database Systems in DC

We are now ready to give a formal model of RTDBS in DC. We will first introduce DC state variables to model the basic primitives of RTDBSs, and then write DC formulas on the introduced state variables to specify various aspects of the systems.

4.1 Basic Model

As said in Section 2, a database consists a set $\mathcal{O}$ of data objects (denoted by x, y, z, etc.) and a set $\mathcal{T} = \{T_i \mid i \leq n\}$ of transactions. Each transaction T_i arrives at the DB system at time λ_i which is unknown in advance. After arriving a transaction performs some *read operations* on some data objects, does some local computations and then performs some *write operations* on some data objects. We assume that the atomic commitment of transactions is ensured. This means that if a transaction has been aborted then it's results will not appear in the database. The model of transactions presented in this paper is a widely accepted one in the literature: each transaction can read and write to a data object at most once during its execution, and reads are before any write.

Now we introduce state variables to capture precisely the behaviour of transactions and data objects, which will be boolean functions of *Time* with *true* being denoted by 1 and *false* being denoted by 0.

Let x be a data object. For each $i \leq n$ a state variable $wr_i(x)$ is introduced to express the behaviour of x. $wr_i(x)$ holds at time t iff the value of x at t is the one written by transaction T_i. By our convention, when a new value is written on an object x at a time t, the object x should hold this value from t to $t + \delta$ for some positive δ (this is reasonable because it takes the data management system a fix amount of time to process a write), and all the objects are initialised by a virtual transaction T_0.

$wr_i \in \mathcal{O} \rightarrow Time \rightarrow \{0, 1\}$
$wr_i(x)(t) = 1$ iff at time t object x holds the value written by T_i

Since a transaction T_i can read a data object x at most once, its view on x can be captured by a state variable $vw_i(x)$. $vw_i(x)$ holds at time t iff T_i has performed a read operation on x successfully before t. At the beginning, T_i has 'no view' on x. Therefore, a read on x is performed at the time that $vw_i(x)$ changes its value from 0 to 1.

$vw_i \in \mathcal{O} \rightarrow Time \rightarrow \{0, 1\}$
$vw_i(x)(t) = 1$ iff T_i has performed a read operation on x successfully before t

A transaction can commit or abort. Therefore, for each $i \leq n$ state variables cm_i and ab_i are introduced to express that T_i has committed or aborted at time t.

$cm_i, ab_i \in Time \rightarrow \{0, 1\}$
$cm_i(t) = 1$ iff T_i has committed successfully before t

$ab_i(t) = 1$ iff T_i has aborted before t

Since we want to specify the execution of the transactions from the beginning to the time that the transactions either commit or abort, we use a state expression $flag \mathrel{\widehat{=}} \bigwedge_{i \leq n} cm_i \vee ab_i$ to express the end of this time interval. The state variable *flag* is true at time t iff for any $i \leq n$ transaction T_i has either committed or aborted at or before t. Thus intervals that express the whole execution of the transaction system should satisfy the following DC formula $\mathcal{E}$:

$$\mathcal{E} \;\;\widehat{=}\;\; \left(\begin{array}{l} \bigwedge_{x \in \mathcal{O}} \lceil wr_0(x) \rceil \wedge \\ \bigwedge_{i \leq n, x \in \mathcal{O}} \lceil \neg vw_i(x) \rceil \end{array} \right) ^\frown true ^\frown \lceil flag \rceil$$

For an interval that satisfies $\mathcal{E}$, at the beginning of the interval, all transactions have neither read anything from nor written anything into the database, and at the end of the interval, all transactions have either committed or aborted.

Fig. 2 represents the state variables introduced so far graphically.

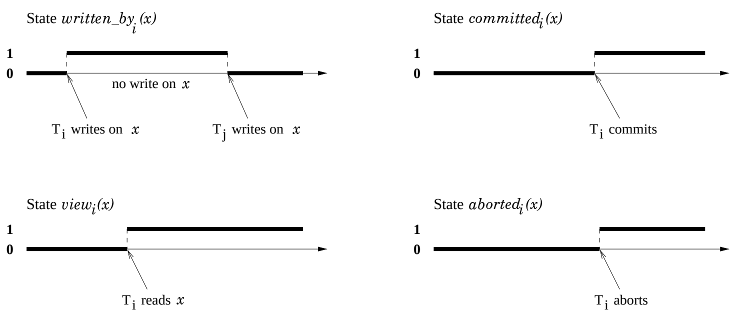

Figure 2. Graphical representation of state variables

Now, we introduce some axioms as DC formulas to express the properties of state variables and their relationships. From now on in this paper, we assume that the operator chop $\frown$ binds more tightly than the connectives $\wedge$ and $\vee$ to avoid the heavy use of brackets.

At any time the value of a data object is given by one and only one transaction:

$$\lceil\!\lceil wr_i(x) \rceil\!\rceil \Rightarrow \bigwedge_{i \neq j} \lceil\!\lceil \neg wr_j(x) \rceil\!\rceil \tag{1}$$

$$\lceil\!\lceil \bigvee_{0 \leq i \leq n} wr_i(x) \rceil\!\rceil \tag{2}$$

As mentioned earlier, for all natural numbers i ($i \leq n$) and for all data objects x the state $vw_i(x)$, cm_i and ab_i can change at most once, and states cm_i and ab_i are mutually exclusive:

$$\lceil\!\lceil vw_i(x) \rceil\!\rceil^\frown true \Rightarrow \lceil\!\lceil vw_i(x) \rceil\!\rceil \tag{3}$$

$$\lceil\!\lceil cm_i \rceil\!\rceil^\frown true \Rightarrow \lceil\!\lceil cm_i \rceil\!\rceil \tag{4}$$

$$\lceil\!\lceil ab_i \rceil\!\rceil^\frown true \Rightarrow \lceil\!\lceil ab_i \rceil\!\rceil \tag{5}$$

$$\lceil\!\lceil cm_i \rceil\!\rceil \Rightarrow \lceil\!\lceil \neg ab_i \rceil\!\rceil \tag{6}$$

From the assumption of atomic commitment it follows that if a transaction has written something into the database then it should commit at the end (of the interval expressing the whole execution of transaction system):

$$\mathcal{E} \Rightarrow (\Diamond \lceil\!\lceil wr_i(x) \rceil\!\rceil \Rightarrow true^\frown \lceil\!\lceil cm_i \rceil\!\rceil) \tag{7}$$

Let *ENV* be the set of the formulas $(1), \ldots, (7)$. *ENV* express the axioms for the state variables introduced so far.

4.2 Serializability

As said in Section 2, in this paper we only consider the conflict serializability which we also call serializability for simplicity. Now we give a characterisation of the serializability in our state model of the databases. As it has been mentioned in Section 2, the serializability says that the relation 'before' between transactions defined by the order of the conflict operations of the execution is a partial ordering on the set of transactions. This idea is formalised as follows.

The order between conflict operations on data object x is defined as follows.

$$\begin{array}{lcl}
WR_{ij}(x) & \widehat{=} & \Diamond \begin{pmatrix} \lceil\!\lceil wr_i(x) \rceil\!\rceil \wedge \\ \lceil\!\lceil \neg vw_j(x) \rceil\!\rceil \end{pmatrix} {}^\frown true^\frown \lceil\!\lceil vw_j(x) \rceil\!\rceil \\
RW_{ij}(x) & \widehat{=} & \Diamond \begin{pmatrix} \lceil\!\lceil vw_i(x) \rceil\!\rceil \wedge \\ \lceil\!\lceil \neg wr_j(x) \rceil\!\rceil \end{pmatrix} {}^\frown true^\frown \lceil\!\lceil wr_j(x) \rceil\!\rceil \\
WW_{ij}(x) & \widehat{=} & \Diamond \lceil\!\lceil wr_i(x) \rceil\!\rceil^\frown true^\frown \lceil\!\lceil wr_j(x) \rceil\!\rceil
\end{array}$$

So, $WR_{ij}(x)$ holds for an interval iff the value of x written by transaction T_i still valid for some time in the interval and transaction T_j performs a read on x after that, and similarly for the others.

Let RW_{ij}, WR_{ij} and WW_{ij} denote the order on the transactions which will not abort, defined by the conflict operations,

$$\begin{array}{lcl}
RW_{ij} & \widehat{=} & \bigvee_{x \in \mathcal{O}} RW_{ij}(x) \wedge TC_{ij} \\
WR_{ij} & \widehat{=} & \bigvee_{x \in \mathcal{O}} WR_{ij}(x) \wedge TC_{ij} \\
WW_{ij} & \widehat{=} & \bigvee_{x \in \mathcal{O}} WW_{ij}(x) \wedge TC_{ij}
\end{array}$$

where

$$TC_{ij} \quad \widehat{=} \quad true^\frown \lceil\!\lceil cm_i \wedge cm_j \rceil\!\rceil$$

Then the relation 'before' defined by the conflict operations for the execution of the transaction system in an interval is the transitive closure of the relations RW_{ij}, WR_{ij} and WW_{ij}. The following DC formula $\mathcal{C}^n_{ij}$ captures the relation 'before': for any i, j, k, $i \neq j \neq k$ let

$$\begin{array}{lcl}
\mathcal{C}^1_{ij} & \widehat{=} & RW_{ij} \vee WR_{ij} \vee WW_{ij} \\
\mathcal{C}^2_{ij} & \widehat{=} & \mathcal{C}^1_{ij} \vee (\mathcal{C}^1_{ik} \wedge \mathcal{C}^1_{kj}) \\
\ldots & & \\
\mathcal{C}^n_{ij} & \widehat{=} & \mathcal{C}^{n-1}_{ij} \vee (\mathcal{C}^{n-1}_{ik} \wedge \mathcal{C}^{n-1}_{kj})
\end{array}$$

Each individual concurrent execution of the transaction system corresponds to exactly an interpretation of state variables. An individual execution is serializable iff it satisfies that the relation 'before' in the interval expressing the whole execution (expressed by $\mathcal{E}$) has no cycle.

Serializability Criterion A concurrent execution of the set of transactions $\mathcal{T}$ is serializable iff it satisfies the following DC formula *SERIAL* for any interval

$$\mathit{SERIAL} \quad \widehat{=} \quad \mathcal{E} \Rightarrow \bigwedge_{i,j\le n, i\ne j} \neg(\mathcal{C}^n_{ij} \wedge \mathcal{C}^n_{ji})$$

4.3 A Case Study: Two-phase Locking Concurrency Control Protocol

In this section, we show the use of our model by giving a formal specification of 2PL-CCP and a formal proof of its correctness.

Recall that in 2PL-CCP, each data object x has a read-lock and a write-lock. A transaction can perform a read on x iff it has a read-lock on x, and a transaction can write to x iff it has a write-lock on x. read-locks can be shared by the transactions, but write-locks cannot. Besides, the read-lock and the write-lock on x are conflicting. This means that if a transaction holds a write-lock on x then another transaction cannot have a read-lock on x and vice-versa.

In order to formalise the protocol, for each $i \le n$, $x \in \mathcal{O}$ we introduce the state variables $rl_i(x)$ and $wl_i(x)$ to express that the read-lock or the write-lock on x is held by the transaction T_i or not at a time.

$rl_i,\ wl_i \in \mathcal{O} \to \mathit{Time} \to \{0,1\}$
$rl_i(x)(t) = 1$ iff transaction T_i holds the read-lock on x at time t
$wl_i(x)(t) = 1$ iff transaction T_i holds the write-lock on x at time t

Transaction executions consist of two phases. In the first phase data object locks are acquired, while in the second phase the data object locks are released and new locks can not be acquired. So, for each transaction T_i we introduce a state variable ph_i to express which phase the transaction T_i is in at a time.

$ph_i \in \mathit{Time} \to \{0,1\}$
$ph_i(t) = 1$ iff transaction T_i is in the obtaining phase

Then, the 2PL-CCP is formalised by the following DC formulas.

The conflicting locks cannot be shared by the transactions. Therefore, for any $i, j \le n, i \ne j, x \in \mathcal{O}$

$$\lceil rl_i(x)\rceil \Rightarrow \lceil \neg wl_j(x)\rceil \tag{8}$$
$$\lceil wl_i(x)\rceil \Rightarrow \lceil \neg rl_j(x)\rceil \wedge \lceil \neg wl_j(x)\rceil \tag{9}$$

Obtaining phase always precedes the releasing phase for any transaction T_i during of its execution (expressed by $\mathcal{E}$).

$$\mathcal{E} \Rightarrow \lceil ph_i\rceil ^\frown \lceil \neg ph_i\rceil \tag{10}$$

A transaction can be in the obtaining phase only if it has not committed or aborted.

$$\lceil ph_i\rceil \Rightarrow \lceil \neg cm_i\rceil \wedge \lceil \neg ab_i\rceil \tag{11}$$

A transaction can obtain locks only in its obtaining phase. Therefore

$$\lceil \neg rl_i(x)\rceil ^\frown \lceil rl_i(x)\rceil \Rightarrow \lceil \neg rl_i(x)\rceil ^\frown \lceil ph_i\rceil ^\frown true \tag{12}$$
$$\lceil \neg wl_i(x)\rceil ^\frown \lceil wl_i(x)\rceil \Rightarrow \lceil \neg wl_i(x)\rceil ^\frown \lceil ph_i\rceil ^\frown true \tag{13}$$

A transaction can release a lock on a data object only in the second phase.

$$\lceil rl_i(x)\rceil ^\frown \lceil \neg rl_i(x)\rceil \Rightarrow \lceil rl_i(x)\rceil ^\frown \lceil \neg ph_i\rceil ^\frown true \tag{14}$$
$$\lceil wl_i(x)\rceil ^\frown \lceil \neg wl_i(x)\rceil \Rightarrow \lceil wl_i(x)\rceil ^\frown \lceil \neg ph_i\rceil ^\frown true \tag{15}$$

A transaction can read or write on a data object only if it holds the corresponding lock on the data object at the time:

$$\lceil \neg vw_i(x)\rceil ^\frown \lceil vw_i(x)\rceil \Rightarrow \Diamond \lceil rl_i(x)\rceil \tag{16}$$
$$\lceil \neg wr_i(x)\rceil ^\frown \lceil wr_i(x)\rceil \Rightarrow \Diamond \lceil wl_i(x)\rceil \tag{17}$$

Let *2PLC* be the set of the DC formulas $(8), \ldots, (17)$ and

$$\mathit{TWOPHASE} \mathrel{\widehat{=}} \bigwedge_{\varphi \in \mathit{2PLC}} \Box\varphi\,.$$

Theorem 1. *(Correctness) With the assumption ENV the 2PL-CCP is correct, i.e. SERIAL is provable from ENV and 2PLC:*

$$\mathit{ENV}, \mathit{2PLC} \vdash \mathit{SERIAL}$$

From the deduction theorem [3] of DC it follows from Theorem 1 that *TWOPHASE* $\Rightarrow$ *SERIAL* under *ENV*, i.e. all executions of the transaction system produced by 2PL-CCP are serializable.

In order to prove the Theorem 1 we need the following lemmas.

Lemma 1. *For any* $m, i, j \le n, i \ne j$

$$\mathit{ENV}, \mathit{2PLC} \ \vdash\ \mathcal{E} \wedge C^m_{ij} \Rightarrow \Diamond \lceil \neg ph_i\rceil ^\frown \lceil ph_j\rceil$$

Lemma 2. *For any* $i, j \le n, i \ne j$

$$\mathit{2PLC} \ \vdash\ \mathcal{E} \wedge \Diamond \lceil \neg ph_i\rceil ^\frown \lceil ph_j\rceil \Rightarrow \neg \Diamond \lceil \neg ph_j\rceil ^\frown \lceil ph_i\rceil$$

Due to space limit the proof of the lemmas is omitted here.

Theorem 1 now is proved as follows.

For any $i, j \le n, i \ne j$

$$\begin{array}{lll} & \mathcal{E} \wedge \mathcal{C}^n_{ij} \wedge \mathcal{C}^n_{ji} \ \Rightarrow & \\ \Rightarrow & \mathcal{E} \wedge \Diamond \lceil \neg ph_i\rceil ^\frown \lceil ph_j\rceil \wedge \Diamond \lceil \neg ph_j\rceil ^\frown \lceil ph_i\rceil & \text{L1, PL} \\ \Rightarrow & false & \text{L2, PL} \end{array}$$

4.4 DC Model for Real-Time Database Systems

In real-time database systems, in addition to the conventional transaction correctness and the logical data consistency criterion, transactions have to meet their deadlines, and the data have to be temporally consistent. In this section we give a formal description for RTDBs and a correctness criterion for the execution of transactions in DC. Before doing so, we need to explain more about data objects and transactions.

In RTDBSs the data objects are divided into groups:

Temporal objects a state of temporal objects may become invalid with the passage of time. Associated with a value of a temporal object is a temporal validity interval. Base temporal objects are those that reflect specific entities in the environment. For simplicity we will consider only base objects in this paper.

Non-temporal objects A non-temporal object is those objects whose state doesn't become invalid with the passage of time.

All transactions, depending on what kind of data objects they can update, are divided into two groups in our model:

Sensor transactions These are transactions which only update base objects. The set of sensor transactions is denoted by $\mathcal{ST}$.

User transactions These are user level transactions with deadlines. They can never update temporal objects. The set of user transactions is denoted by $\mathcal{T}$.

Our model of RTDBS is an extension of the model described in Section 4.1. In addition to set $\mathcal{O}$ of data objects we introduce set $\mathcal{TO}$ of temporal data objects (denoted by X, Y, Z, etc.). Actually, at each moment of time a temporal data object is represented by several versions which are valid for some time interval. We assume that the system is observed on a finite interval. Hence there is only a finite number of versions of a temporal data object X (denote this number as M). Note that at the same moment of time there may be several versions of same temporal object in database that are valid. The old (not fresh) versions of temporal object are deleted according to the specific politics applied in the DB.

The state variables to capture the behaviour of temporal data objects are as follows.

Let X be a temporal data object. For each non-negative integer $q \leq M$ there is a non-temporal data object (X, q) and a state variable $\mathit{vr}_q(X)$ to reflect the q'th version for the value of X. $\mathit{vr}_q(X)$ holds at time t iff q'th version of data object X has been created (before time t) and still valid at time t.

$\mathit{vr}_q \in \mathcal{TO} \rightarrow \mathit{Time} \rightarrow \{0,1\}$
$\mathit{vr}_q(X)(t) = 1$ iff t is in the valid interval of the q'th version of X

There is a positive lower bound δ' for the valid interval (depending on the sampling periods), and each version may have only a single interval of validity. Therefore,

$$\lceil\neg \mathit{vr}_q(X)\rceil^\frown\lceil \mathit{vr}_q(X)\rceil^\frown\lceil\neg \mathit{vr}_q(X)\rceil \;\Rightarrow\; \ell \geq \delta' \quad (18)$$

$$\begin{array}{l}\lceil \mathit{vr}_q(X)\rceil^\frown \mathit{true} \;\Rightarrow \\ \lceil \mathit{vr}_q(X)\rceil \vee \lceil \mathit{vr}_q(X)\rceil^\frown\lceil\neg \mathit{vr}_q(X)\rceil\end{array} \quad (19)$$

Data read by transactions must be not only fresh, but also temporally correlated. This leads to the notion of relative consistency.

The relative consistency says that data objects from some data set should be temporally correlated. Any set R of versions of temporal data objects, i.e. R is a set of (X, q), is associated with the length of its *relative validity interval* denoted by $\mathit{rvi}(R)$. The set R of data read by a transaction must be relative consistent, which means that the time distance between their creation is not more than $\mathit{rvi}(R)$.

The relative consistency of a set R of versions is now expressed by the following DC formula $\mathit{RCONS}(R)$, meaning that R is relatively consistent iff DC formula $\mathit{RCONS}(R)$ is true for all intervals.

$$\mathit{RCONS}(R) \mathrel{\widehat{=}} \bigwedge_{(X,q),(Y,r)\in R} \Box\begin{pmatrix}\lceil \mathit{vr}_q(X) \wedge \neg \mathit{vr}_r(Y)\rceil^\frown\lceil \mathit{vr}_r(Y)\rceil \\ \Rightarrow\;(\ell \leq \mathit{rvi}(R))^\frown\lceil \mathit{vr}_r(Y)\rceil\end{pmatrix} \quad (20)$$

The example of relatively consistent set (of two data objects) is shown in the Figure 3.

From the property of the sensor transactions, we can identify them with the temporal objects that they write to.

Every user transaction T_i is associated with a deadline d_i.

In our model, in each execution each transaction $T_i \in \mathcal{T}$ is associated with a set R_i of versions of temporal data objects which are read by it. Therefore, a transaction in a real-time database can commit iff

1. it is logically consistent
2. it meets its deadline, and
3. it reads temporally consistent sets of data, and the data it reads are still valid when it commits.

In order to express this as a DC formula we introduce for each transaction T_i a state variable ar_i with the following meaning:

$\mathit{ar}_i \in \mathit{Time} \rightarrow \{0,1\}$
$\mathit{ar}_i(t) = 1$ iff at time t transaction T_i is in the system and has not been committed or aborted

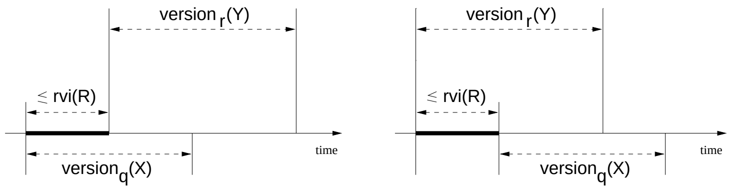

Figure 3. Relative consistency

So, transaction T_i can be committed only if it meets its deadline and reads a valid version of data objects at the time. Therefore, the following DC formula CM_i is satisfied: where

$$CM_i \mathrel{\widehat{=}} \lceil\neg ar_i\rceil^\frown\lceil ar_i\rceil^\frown\lceil cm_i\rceil \Rightarrow RCONS(R_i) \wedge AR_i$$

$$AR_i \mathrel{\widehat{=}}$$

$$\Box\left(\lceil ar_i\rceil \Rightarrow \begin{pmatrix}\ell \le d_i \wedge \\ \bigwedge_{(X,q)\in R_i}\lceil vw_i(X,q)\rceil \Rightarrow \lceil vr_q(X)\rceil\end{pmatrix}\right)$$

Let $CM \mathrel{\widehat{=}} \bigwedge_{i\le n} CM_i$

Current research (see, e.g. [12]) shows that it is not necessary for the concurrency control protocols to deal with sensor transactions. So, the concurrency control is needed only for user transactions (from set $\mathcal{T}$). Now we are ready to give formal description of correctness criterion of execution of transactions in RTDBS as conjunction of the correctness criterion for the untimed case (presented in Section 4.2 by formula *SERIAL*) and the time constraints for the transactions and their read data (expressed by formula *CM*).

Correctness Criterion of execution of transactions in RTDBS *An execution of set $\mathcal{T}$ of transactions is correct iff for any interval it satisfies formulas SERIAL $\wedge$ CM.*

5 Conclusion

In this work we have presented a formal description of real-time database systems and the correctness criterion for the executions of a transaction system. We have specified and verified formally the two phase locking concurrency control protocol using the proof system of DC. The proof can be checked easily by a DC proof-checker. With our correctness criterion, we can decide the feasibility of a real-time transaction system, which is the satisfiability of the DC formula for the correctness criterion with the given parameters for the transaction deadline and the validity of the temporal data, using the model-checking tool of DC.

We find that formalisation of RTDBSs using DC is quite natural and intuitive. Especially, in RTDBS we have to deal with the schedulers (have not been covered in this paper) in which the concept duration must be used for specifying the preemption of the processors, the use of DC could be inevitable.

References

[1] A. Bestavros, K.-J. Lin, and S. H. Son. *Real–Time Database Systems: Issues and Applications*. Kluwer Academic Publishers, 1997.

[2] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

[3] M. R. Hansen and Z. Chaochen. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9:283–330, 1997.

[4] D. V. Hung. Modelling and verification of biphase mark protocols in Duration Calculus using PVS/DC. In *Proceedings of the 1998 IEEE International Conference on Application of Concurrency to System Design (CSD'98)*, pages 88–98, Aizu-wakamatsu, Fukushima, Japan, Mar. 1998.

[5] P. Krzyżagórski and T. Morzy. Optimistic concurrency control algorithm with dynamic serialization adjustment for firm deadline real-time database systems. In *Proceedings of the Second International Workshop on Advances in Databases and Information Systems (ADBIS"95)*, volume 1, pages 21–28, June 1995.

[6] K. Lam, S. H. Son, and S. Hung. A priority ceiling protocol with dynamic adjustment of serialization order. In *13th IEEE Conference on Data Engineering (ICDE'97)*, Birmingham, UK, Apr. 1997.

[7] S. H. Son, S. Park, and Y. Lin. An integrated real-time locking protocol. In *Eighth IEEE International Conference on Data Engineering*, pages 527–534, Phoenix, Arizona, Feb. 1992.

[8] J. Stankovic, S. H. Son, and J. Hansson. Misconceptions About Real-Time Databases. *IEEE Computer*, 32(6):29–36, June 1999.

[9] A. Thomasian. Concurrency control: Methods, performance, and analisis. *ACM Computing Surveys*, 30(1):70–119, Mar. 1998.

[10] J. D. Ullman. *Principles Of Database And Knowledge-Base Systems*. W.H. Freeman & Company, 1988.

[11] M. Xiong, R. Sivasankaran, J. A. Stankovic, K. Ramamritham, and D. Towsley. Scheduling transactions with temporal constraints: Exploiting data semantics. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, Washington, DC, Dec. 1996.

[12] M. Xiong, R. Sivasankaran, J. A. Stankovic, K. Ramamritham, and D. Towsley. Scheduling access to temporal data in real-time databases. In *Real–Time Database Systems: Issues and Applications*, pages 167–192. Kluwer Academic Publishers, 1997.