Annales Univ. Sci. Budapest., Sect. Comp. 17 (1998) 49-64

CORRECTNESS CRITERIA FOR DATABASES USING ABSTRACT DATA TYPES

A. Benczúr and Chu Ky Quang (Budapest, Hungary)

1. Introduction

The concurrency control problem for database systems is traditionally investigated using a simple model for transactions in which users are restricted to access the database by uninterpreted read and write operations. A concurrent execution, or schedule, of a set of transactions is called *serializable* if it is equivalent to a serial schedule. "Equivalent" means that neither the transactions nor the database can tell the difference: every transaction gets the same view of the data, and the final state of the database becomes the same.

Schedulers based on such a model allow only limited concurrency because they do not have any information on the meaning of the operations. Researchers working nowadays on this problem are interested in transaction models using more semantic information to increase the amount of concurrency allowed by schedulers. The work in this area includes widely different approaches such as enriching the read/write model with additional operations, using models based on abstract data types, using semantic information provided by the users, etc. In this paper we give a model capturing the semantics of operations on abstract data type objects through their specifications.

We start by defining the specification of the operations, which reflects the effect of the execution of the operations on the database and for the users. A transaction is viewed as a sequence of operations, and a schedule of a set of transactions is an interleaving sequence of all the operations occurring in the transactions. Serializability of a schedule is defined entirely semantically: a schedule of a set of transactions is serializable if it has the same effect on the database and for the users as the execution of the transactions in some serial order. We use this serializability as the cornerstone of database correctness

Supported by the Hungarian National Science Grant (OTKA), grant Nr. 2149.

criteria. We first look at the complexity of testing serializability and show that it is NP-complete. However, we show an infinite sequence of subclasses of serializable schedules whose serializability can be tested by powerful polynomial time algorithms. The approximate algorithms are obtained by restricting, in various ways, the "amount" of context of the schedule examined at a time.

The idea of using the semantics of operations to increase concurrency is not entirely new. However, no formal presentation of these ideas has previously been available. Many of our results are similar to those in [8], where authors present them in a transaction model using relational database updates.

The remainder of this paper is organized as follows: In Section 2 we present our formal model, the serializability criterion and the complexity of testing serializability. In Section 3 we define left-commutativity and use it to show polynomial approximations of serializability. Finally, in Section 4 we make some conclusions and give some suggestions for further work.

2. The model

In this section we formally present our transaction model, serializability criterion and the complexity of testing the criterion.

2.1. Basic notions

The database consists of a finite set of *objects* of *abstract data types*. Every object is always in one of its possible states. We denote the objects by letters $x, y, z \ldots$ A *state of the database* is the set of states of all the objects in the system. A *consistent state* of the database is a state which satisfies the constraints imposed on the database. The initial database state is supposed to be a consistent state.

A *database operation* is expressed by a tuple [x, *op*, *inp*] with the meaning: *op* is the operation on object x with input parameter *inp*. We suppose that database operations are atomic in the following sense: during the execution of an operation no other action is also performed concurrently. By that assumption we can exploit some properties of the operations through their specification. The specification of operations will be defined below.

A *transaction* is a sequence of database operations. Each transaction must satisfy the consistency property: if it is executed alone in a consistent state, it leaves the database in a consistent state.

A *schedule* u of set of transactions $\{t_1, ..., t_n\}$ is an interleaving sequence of operations originated from the transactions, such that for every i the restrictions $u_{|t_i} = t_i$, where the restriction $u_{|t_i}$ is the subsequence of u having only the operations of t_i. The schedule u of transactions $\{t_1, ..., t_n\}$ is *serial* if, for any two different transactions t_i, t_j, either all the operations of t_i precede all operations of t_j, or vice versa.

Deciding whether a concurrent execution of transactions in a database is correct is examined from the point of view of the database and that of the users. Hence our definition of specification of operation $[x,\ op,\ inp]$ must reflect the execution's effect on the database as well as for the users.

Definition 1. Let $a = [x,\ op,\ inp]$ be a database operation. Its specification is the following tuple (Q, R, tr_a, rv_a), where Q is the state set of object x, R is the set of return values of the operation, tr_a and rv_a are state transition and return value mappings as follows: $tr_a : Q \to Q$, $rv_a : Q \to R$.

The following example illustrates this concept.

Example 1. Let BA be a bank account object, whose state set is $\Re$, the set of positive reals. On object BA we define operations di=[BA, *deposit*, i] which deposits an amount i to BA, wi=[BA, *withdraw*, i] which withdraws an amount i from BA and b=[BA, *balance*, -] which examines the balance of BA. The specifications of operations are as follows:

- The specification of di is $(\Re,\ \{ok\},\ tr_{di},\ rv_{di})$, where $tr_{di}(j) = j + i$ and $rv_{di}(j) = ok$ for every $j \in \Re$;
- The specification of wi is $(\Re,\ \{ok,\ no\},\ tr_{wi},\ rv_{wi})$, where $tr_{wi}(j) = j - i$, $rv_{wi}(j) = ok$ if $j \geq i$; and $tr_{wi}(j) = j$, $rv_{wi}(j) = no$ if $j < i$;
- The specification of b is $(\Re,\ \Re,\ tr_b,\ rv_b)$, where $tr_b(j) = rv_b(j) = j$ for every $j \in \Re$.

Let $a_1, ..., a_n$ be database operations on the same object x, where the specification of operation a_i is $(Q, R_i, tr_{a_i}, rv_{a_i})$ $(i = 1, ..., n)$. We say that the *sequence* $a_1...a_n$ *is defined* at $s \in Q$ if mappings tr_{a_1}, rv_{a_1} are defined at s and for every i $(i = 2, ..., n)$, mappings tr_{a_i}, rv_{a_i} are defined at the state $tr_{a_{i-1}}(...(tr_{a_1}(s))...)$. In that case we denote $tr_{a_n}(tr_{a_{n-1}}...(tr_{a_1}(s))...)$ with $tr_{a_1...a_n}(s)$.

For the above operations $a_1, ..., a_n$, let $a_{i_1}...a_{i_n}$ be a permutation of $a_1...a_n$. We write $a_1...a_n \approx a_{i_1}...a_{i_n}$ if and only if for every state s of x, where the sequence $a_1...a_n$ is defined, $a_{i_1}...a_{i_n}$ is also defined and the following two conditions hold:

(1) $tr_{a_1...a_n}(s) = tr_{a_{i_1}...a_{i_n}}(s)$;

(2) if $a_j = a_{i_k}$, then $rv_{a_j}(tr_{a_1...a_{j-1}}(s)) = rv_{a_{i_k}}(tr_{a_{i_1}...a_{i_{k-1}}}(s))$.

Let u, u' be two schedules of the same transactions. We say that *schedule u is reducible to schedule u'* (u a u') if and only if for every object x of database we have $u_{|x} \approx u'_{|x}$.

We note that relations f and a are reflexive, transitive, but not symmetric. If u is reducible to u', then on all states of the database where schedule u is defined, the schedules u and u' are "equivalent".

Definition 2. A schedule u is serializable if and only if there exists a serial schedule u_s of the same transactions such that $u = u_s$. The class of all serializable schedules is denoted by SER.

From the consistency property of every transaction, each schedule in SER is correct on all states of the database where it is defined.

2.2. Computational complexity of class SER

In this section we investigate the complexity of checking the serializability of a given schedule by using the following assumption.

Assumption. For any given schedules u and u', testing whether u is reducible to u' is always done by a polynomial algorithm by using the specifications of the database operations.

By that assumption, testing if a schedule u of a set $\{t_1, ..., t_n\}$ of transactions is serializable can be done by trying out all permutations $t_{i_1}...t_{i_n}$ of sequence $t_1...t_n$ and checking whether u a $t_{i_1}...t_{i_n}$. Unfortunately, there is no significantly better testing algorithm. In order to prove this we reduce the problem of testing serializability of a schedule in read/write model to our problem. NP-complete property of that problem is proved by Papadimitriou in [6].

We now introduce the notions and results related to the problem in [6]. A database consists of a finite set of variables. A *two-step transaction* is a sequence of read and write operations where all read operations must precede all writes. A *history* h of two-step transactions $\{t_1, ..., t_n\}$ is an interleaving sequence of operations originated from that transactions such that for every i the restrictions $h_{|t_i} = t_i$ hold. Two histories of the same two-step transactions are *equivalent* if and only if given a set of initial values for the variables, and any local computations of the transactions, the values of the variables are identical after the execution of both histories. A history is *serializable* if it is equivalent to a serial history of the same transactions.

Let h be a history of two-step transactions $\{t_1, ..., t_n\}$ and $\{x_1, ..., x_m\}$ be the set of all variables appearing in h. The *augmentation* of history h is the sequence $\underline{h} = t_o.h.t_f$, where initial transaction t_o writes every variable, reading none, and final transaction t_f reads every variable, writing none. We say that $R(x)$ *reads* x *from* $W(x)$ in $\underline{h}$ if the sequence $\underline{h}$ is in the form: $...W(x)\alpha R(x)...$, where α does not consist of any $W(x)$. The definition of a *live* transaction in $\underline{h}$ is as follows:

(a) t_f is live in $\underline{h}$.

(b) If for some live transaction t_j, $R(x)$ of t_j reads x from a $W(x)$ of t_i in $\underline{h}$, then t_i is also live in $\underline{h}$.

(c) All the live transactions in $\underline{h}$ are defined by (a) and (b) above.

Given any history h of two-step transactions $\{t_1, ..., t_n\}$, we are going to define a *polygraph* $P(h) = (T, A, B)$. T is the set of live transactions in h. First, set A contains the arcs $\{(t_o, t) \mid t \in T \setminus \{t_o\}\} \cup \{(t, t_f) \mid t \in T \setminus \{t_f\}\}$. Second, whenever a $R(x)$ of transaction t_j reads variable x from $W(x)$ of t_i in $\underline{h}$, we add the arc (t_i, t_j) in A. Furthermore, if there is a $W(x)$ of a third transaction t_p, then we add the arc pair $((t_p, t_i), (t_j, t_p))$ in B. A polygraph is *acyclic* if there is some series of choices of one arc from each arc pair that results in an acyclic graph in the ordinary sense.

Lemma 1. *Two histories h and h' of the same two-step transactions are equivalent if and only if they have the same set of live transactions, and a R(x) of a live transaction reads x from a W(x) in* $\underline{h}$ *if and only if the R(x) also reads x from the W(x) in* $\underline{h'}$.

Lemma 2. *A history h without dead transactions is serializable if and only if P(h) is acyclic.*

Lemma 3. *Testing whether a history h is serializable is NP-complete, even if h has no dead transactions.*

We now state the result about complexity of testing serializability of a schedule in our model.

Theorem 1. *Deciding whether a schedule u of a set of transactions over the database is serializable is a NP-complete problem.*

Proof. The set of serializable schedules is defined in NP, since to show that u is serializable, one only needs to construct a serial schedule u_s and check by a polynomial algorithm that $u\ a\ u_s$ (existence of that algorithm is ensured by the above assumption). We will show next that a known NP-complete problem, the problem of Lemma 3 above (for convenience we will call it problem 1) reduces to our problem (called problem 2).

A tuple $(DB, h, \{t_1, ..., t_n\})$ can be viewed as an input of problem 1, where DB is a database consisting of variables, h is a history of live two-step transactions $\{t_1, ..., t_n\}$. We will construct a polynomial algorithm that maps an input $(DB, h, \{t_1, ..., t_n\})$ of problem 1 to an input $(DB', u, \{t'_o, t'_1, ..., t'_n, t'_f\})$

of problem 2, where DB' is a database consisting of objects, u is a schedule of transactions $\{t'_o, t'_1, \ldots, t'_n, t'_f\}$ such that history h is serializable in read/write model if and only if schedule u is serializable in our model.

We denote the set of nonnegative integers by $\aleph$. To each variable x of the database DB we associate an object x' of DB'. While domain for variable x is any set, state set of x' is fixed by $\aleph$.

The initial transaction t'_o is the sequence of operations $Wx'0 = [x',\ \mathit{write},\ 0]$ (for every object x'). The specification of $Wx'0$ is tuple $(\aleph,\ \{\mathit{null}\},\ tr_{Wx'0},\ rv_{Wx'0})$, where $tr_{Wx'0}(s) = 0$ and $rv_{Wx'0}(s) = \mathit{null}$ $(\forall s \in \aleph)$.

The final transaction t'_f is the sequence of operations $Rx' = [x',\ \mathit{read},\ -]$ (for every object x'). The specification of Rx' is $(\aleph,\ \aleph,\ tr_{Rx'}, rv_{Rx'})$, where mappings $tr_{Rx'}(s) = rv_{Rx'}(s) = s$ $(\forall s \in \aleph)$. We note that the final transaction t'_f does not change the state of database DB' and its return values are the state of DB'.

The algorithm constructing schedule u of transactions $\{t'_o, t'_1, \ldots, t'_n, t'_f\}$ from history h of transactions $\{t_1, \ldots, t_n\}$ works as follows. At first, the algorithm assigns $u = t'_o$. Next, the algorithm will see the operations of h in the order from the beginning to the end along history h. Suppose that the current operation is O of transaction t_i, then the current schedule u is lengthened by the operation *map*(O) of transaction t'_i, i.e. $u = u.\mathit{map}(O)$. Operation *map*(O) is defined by the following rules:

(a) If the operation O is $R(x)$, then $\mathit{map}(R(x)) = Rx' = [x',\ \mathit{read},\ -]$ with the specification $(\aleph,\ \aleph,\ tr_{Rx'}, rv_{Rx'})$, where mappings $tr_{Rx'}(s) = rv_{Rx'}(s) = s$ $(\forall s \in \aleph)$.

(b) If O is the k-th $W(x)$ operation in h, then $\mathit{map}(W(x)) = Wx'k = [x',\ \mathit{write},\ k]$ with the specification $(\aleph,\ \{\mathit{null}\},\ tr_{Wx'k},\ rv_{Wx'k})$, where $tr_{Wx'k}(s) = k$ and $rv_{Wx'k}(s) = \mathit{null}$ $(\forall s \in \aleph)$.

At last, the algorithm lengthens u by $t'f$, i.e. $u = u.t'f$. Clearly, the computational complexity of the algorithm is a linear function of the length of history h. We now prove that history h is serializable in the read/write model if and only if schedule u is serializable in our model.

(Only if). Suppose that two histories h and $h_s = t_{i_1} \ldots t_{i_n}$ are equivalent where $t_{i_1} \ldots t_{i_n}$ is a permutation of $t_1 \ldots t_n$. Let $\underline{h}$ and $\underline{h}_s$ be the augmentations of h and h_s, respectively. We will prove that $u\ a\ u_s = t'_o t'_1 \ldots t'_n t'_f$. Since the two schedules u and u_s are defined at every state of database DB', and since the final transaction t'_f is at the end of both schedules, we only need to show that for any initial state of DB', the two return values of each operation in u and u_s are identical. Since the return value of every $Wx'k$ operation is "*null*", we only need to examine the Rx' operations.

Given an Rx' operation in schedule u, suppose that its return value is k. Hence, there must be $Wx'k$ operation preceding Rx' in u; and there are no $Wx'k_1$ ($k_1 \neq k$) between them. Therefore, $R(x)$ reads x from $W(x)$ in history $\underline{h}$, where $map(R(x)) = Rx'$ and $map(W(x)) = Wx'k$. Since the two-step transactions in history h are live, by Lemma 1, the $R(x)$ also reads x from $W(x)$ in $\underline{h}_s$. Thus, in schedule u_s the $Wx'k$ precedes the Rx'; and there are no $Wx'k_1$ ($k_1 \neq k$) between them. We conclude that the return value of Rx' is also k in u_s.

(If). Suppose that $u = u_s$, where u_s is a serial schedule of transactions $\{t'_o, t'_1, ..., t'_n, t'_f\}$. Let $P(h) = (T, A, B)$ be the polygraph of history h. We now prove that $P(h)$ is acyclic.

Given an arc $(t_i, t_j) \in A$, by the definition of $P(h)$ an operation $R(x)$ of t_j reads x from an operation $W(x)$ of t_i in $\underline{h}$. Suppose that the $W(x)$ is the k-th write operation in history h. Let $Rx' = map(R(x))$ and $Wx'k = map(W(x))$ be the operations of transactions t'_j and t'_i, respectively. Clearly, the return value of the Rx' is k. Since $u = u_s$, the return values of Rx' in both schedules u and u_s must be equal to k. Since $Wx'k$ is the unique operation writing k to the object x', then t'_i must precede t'_j in the serial schedule u_s.

Suppose that in $P(h)$ for the arc $(t_i, t_j) \in A$ we also have an arc pair $((t_p, t_i), (t_j, t_p)) \in B$ in consequence of a write operation $W(x)$ of t_p. We suppose that the $W(x)$ of t_p is the k_1-th write operation in history h. Let $Wx'k_1 = map(W(x))$ be the operation of transaction t'_p in schedule u. Since $k \neq k_1$, t'_p cannot appear between t'_i and t'_j in the serial schedule u_s. We pick arc (t_p, t_i) from the pair if t'_p precedes t'_i in the serial schedule u_s, and pick (t_j, t_p) otherwise. The linear order of serial schedule u_s will be consistent with the arcs in A and the arcs chosen from the pairs in B. We conclude that $P(h)$ is acyclic. By Lemma 2 the history h is serializable.

One way to cope with the above complexity result is to look for restricted notions of serializability, which are decidable in polynomial time. In the following section, by exploiting different semantics of operations, we present a series of such restrictions.

3. Polynomial approximations of serializability

First we introduce the left-commutability relation between sequences of database operations and then we exploit it to show subclasses of serializable schedules that can be recognized by polynomial algorithms.

3.1. Left-commutability between sequences of database operations

Definition 3. Let α, β be sequences of database operations, possibly on different objects. We say that β is left-commutable with α if and only if for every object x either at least one of the two sequences $\alpha_{|x}$ and $\beta_{|x}$ is empty, or both $\alpha_{|x}$ and $\beta_{|x}$ are not empty and $\alpha\beta_{|x} \approx \beta\alpha_{|x}$. Otherwise, if there is an object x such that both $\alpha_{|x}$ and $\beta_{|x}$ are not empty, but $\alpha\beta_{|x} \approx \beta\alpha_{|x}$ is not satisfied, we say that β left-conflicts with α.

To illustrate left-commutability, we will see the following example.

Example 2. Given the object bank account BA introduced in Example 3.1, the following table shows left-commutability between the operations *deposit, withdraw* and *balance. "Yes"* indicates that the operation for the given row is left-commutable with the operation for the column, otherwise, *"no"* indicates that the operation for the given row left-conflicts with the operation for the column.

	[BA, deposit, j]	[BA, withdraw, j]	[BA, balance, -]
[BA, deposit, i]	yes	no	no
[BA, withdraw, i]	no	no	no
[BA, balance, -]	no	no	yes

Directly from the above definition, we state the following lemmas.

Lemma 4. *Let $\alpha, \beta, \gamma_1, \gamma_2$ be sequences of operations. If β is left-commutable with α, then for every object x, $\gamma_1\alpha\beta\gamma_{2|x} \approx \gamma_1\beta\alpha\gamma_{2|x}$.*

Proof. Let x be an object. If at least one of $\alpha_{|x}$ and $\beta_{|x}$ is empty, for example $\alpha_{|x}$ is empty, then $\gamma_1\alpha\beta\gamma_{2|x} = \gamma_1\beta\alpha\gamma_{2|x} = \gamma_1\beta\gamma_{2|x}$. Since the relation $\approx$ is reflexive, we have $\gamma_1\alpha\beta\gamma_{2|x} \approx \gamma_1\beta\alpha\gamma_{2|x}$.

Otherwise, both $\alpha_{|x}$ and $\beta_{|x}$ are not empty, let s be a state of x such that $\gamma_1\alpha\beta\gamma_{2|x}$ is defined at s. Since β is left-commutable with α and $\alpha\beta_{|x}$ is defined at $s_1 = tr_{\gamma 1|x}(s)$, then $\alpha\beta_{|x} \sim_{s1} \beta\alpha_{|x}$. Thus, $\gamma_1\alpha\beta\gamma_{2|x} \sim_s \gamma_1\beta\alpha\gamma_{2|x}$. We conclude $\gamma_1\alpha\beta\gamma_{2|x} \approx \gamma_1\beta\alpha\gamma_{2|x}$.

Lemma 5. *Given two sequences α, β, of operations, $\alpha = a_1 \ldots a_p$ and $\beta = b_1 \ldots b_q$ such that for every $i = 1, \ldots, q$ and $j = 1, \ldots, p$, b_i is left-commutable with a_j. Then β is left-commutable with α.*

Proof. Given an object x, suppose that $\alpha_{|x} = a_{i_1}...a_{i_{p'}}$ and $\beta_{|x} = b_{j_1}...b_{j_{q'}}$. Since each b_i is left-commutable with each a_j, by Lemma 4, we have $\alpha\beta_{|x} = = a_{i_1}...a_{i_{p'}}b_{j_1}...b_{j_{q'}} \approx a_{i_1}...b_{j_1}a_{i_{p'}}...b_{j_{q'}} \approx ... \approx b_{j_1}...b_{jq'}a_{i_1}...a_{i_{p'}} = \beta\alpha_{|x}$.

Assumption. We suppose that for any two sequences α and β of operations testing, whether β is left-commutable with α, is always done by a polynomial algorithm using specification of database operations.

Using this assumption, we now introduce subclasses of serializable schedules, which can be recognized in polynomial time.

3.2. Conflict-serializable schedules

Let u be a schedule of transactions $\{t_1, ..., t_n\}$. The *conflict-serialization graph* of schedule u is $G_c(u) = (T, A)$, where $T = \{t_1, ..., t_n\}$, $(t_i, t_j) \in A$ if and only if there is an operation a of t_i and an operation b of t_j such that b left-conflicts with a and a precedes b in u.

Definition 4. Schedule u is conflict-serializable if and only if the graph $G_c(u)$ is acyclic.

The class of conflict-serializable schedules is denoted by *CSER*. By the above assumption, the conflict-serialization graphs can be constructed by polynomial algorithms. Beside that, acyclicity of graphs can also be recognized by polynomial algorithms. Therefore, conflict-serializable schedules can be recognized by polynomial algorithms. Recall that the class of serializable schedules is denoted by *SER*, correctness of conflict-serializable schedules is proved by the following theorem.

Theorem 2. *We have* $CSER \subset SER$.

Proof. We first prove $CSER \subseteq SER$. Let u be a schedule of transactions $\{t_1, ..., t_n\}$ such that graph $G_c(u) = (T, A)$ is acyclic. Since the graph is acyclic, we can sort the transactions in T in a linear order u_s consistent with A. Without loss of generality, we assume $u_s = t_1t_2...t_n$. We prove that $u = t_1t_2...t_n$.

Let a be the first operation of t_1, we now consider its position in u, suppose that $u = u_1bau_2$. Since a is the first operation of t_1, b must be an operation of a transaction $t_i \neq t_1$. Since the linear order u_s is consistent with A, $(t_i, t_1) \notin A$. By the definition of graph $G(u)$, a must be left-commutable with b. By Lemma 4, for every object x, we have $u_{|x} = u_1bau_{2|x} \approx u_1abu_{2|x}$. By the definition of a, we have $u = u_1bau_2 = u_1abu_2$.

We now consider again the position of operation a in u_1abu_2. By the same deduction, we can move the operation a to the left. Since relation a is transitive, we finally have $u = u_1bau_2 = u_1abu_2 = ... = au_1bu_2$.

For the second operation in $u_s = t_1t_2...t_n$, we consider the operation in au_1bu_2, similarly, we move it to the second position. Considering consecutively from left to right every operation in $u_s = t_1t_2...t_n$, at last we can conclude that $u = t_1t_2...t_n$.

Strictness of the inclusion is proved by showing a schedule u so that $u \in$ $\in SER$, but $u \neq CSER$. We consider the schedule u of transactions $\{t_1, t_2, t_3\}$ on object bank account BA given in Example 1, where operations with index i are of transaction t_i.

$$u =$$

$$= [BA,\ deposit,\ 100]_2[BA,\ withdraw,\ 50]_3[BA, deposit, 10]_1[BA,\ balance,\ -]_3$$

It can be seen easily that $u = t_2t_1t_3$, which means $u \in SER$. Using left-commutability between the operations given in Example 2, graph $G_c(u)$ has a cycle $t_1 \rightarrow t_3 \rightarrow t_1$. Thus, we conclude $u \notin CSER$.

In some sense, conflict-serializability is the most restricted notion of serializability because it takes into account only conflicts between two individual operations, without any regard for the context. At the other extreme, serializability takes into account the entire context. We next exhibit intermediate-serializability and k-serializability looking at something from the context in various ways.

3.3. Intermediate-serializable schedules

Intuitively, intermediate-serializability takes into account conflicts detectable by looking at individual operations occurring in a schedule and the prefixes of transactions occurring in the schedule before that operation.

Let u be a schedule of transactions $\{t_1, ..., t_n\}$. The *intermediate-serialization graph* of schedule u is $G_I(u) = (T, B)$, where $T = \{t_1, ..., t_n\}$ and $(t_i, t_j) \in B$ if and only if there is an operation b of t_j such that $u = u_1bu_2$ and b left-conflicts with $u_{1|t_i}$.

Definition 5. Schedule u is intermediate-serializable if and only if the graph $G_I(u)$ is acyclic.

The class of intermediate-serializable schedules is denoted by *ISER*. Similarly as the class *CSER*, the class *ISER* can be recognized by polynomial algorithms. We now show the relationship between classes *CSER, ISER* and *SER*.

Theorem 3. *The following statements hold:*

(1) $ISER \subset SER$,

(2) $CSER \subset ISER$.

Proof. (1) $ISER \subset SER$. We first prove $ISER \subseteq SER$. Let u be a schedule of transactions $\{t_1, ..., t_n\}$ such that $u \in ISER$, then graph $G_I(u) = = (T, B)$ is acyclic. Hence, we can sort the transactions in T in a linear order u_s consistent with B. Without loss of generality, assume that $u_s = t_1t_2...t_n$, and we will prove that $u = t_1t_2...t_n$ (i.e. $u \in SER$).

If $u \neq t_1t_2...t_n$, then there is an operation $a_k \in t_k$ so that the sequence of the operations preceding it in u is a sequence of prefixes of the transactions in an order consistent with u_s : $u = \alpha_{i_1}...\alpha_{i_p}\alpha_{i_{p+1}}...\alpha_{i_q}a_k u_1$, each α_{i_j} is a prefix of transaction t_{i_j} and $i_1 < ... < i_p \leq k < i_{p+1} < ... < i_q$. The linear order u_s consistent with B follows that for every $j = p+1, ..., q$, (t_{i_j}, t_k) is not in B. It leads to the conclusion that a_k is left-commutable with α_{i_j} (for $j = p+1, ..., q$). By Lemma 4 and the definition of relation a, we have $u = \alpha_{i_1}...\alpha_{i_p}a_k\alpha_{i_{p+1}}...\alpha_{i_q}u_1$.

By the same deduction for $\alpha_{i_1}...\alpha_{i_p}a_k\alpha_{i_{p+1}}...\alpha_{i_q}u_1$, at last we sort the operations of that sequence in the order u_s. By the transitivity of a, we conclude $u = t_1t_2...t_n$.

To prove $ISER \subset SER$, we show that the schedule $u \in SER$ in the proof of Theorem 2 is not in $ISER$: $u \notin ISER$. We rewrite the schedule u of transactions $\{t_1, t_2, t_3\}$ as follows:

$$u =$$

$[BA,\ \textit{deposit},\ 100]_2[BA,\ \textit{withdraw},\ 50]_3[BA,\ \textit{deposit},\ 10]_1[BA,\ \textit{balance},\ -]_3$.

Using left-commutability between operations given in Example 2, the graph $G_I(u)$ has a cycle $t_1 \to t_3 \to t_1$. Thus, we conclude $u \notin ISER$.

(2) $CSER \subset ISER$. Let $G_c(u) = (T, A)$ and $G_I(u) = (T, B)$ be conflict-serialization and intermediate-serialization graphs, respectively of a schedule u of set $T = \{t_1, ..., t_n\}$ of transactions. To prove $CSER \subseteq ISER$, we will prove $B \subseteq A$, then we can conclude that from acyclicity of $G_c(u)$ follows acyclicity of $G_I(u)$.

Given an arc $(t_i, t_j) \in B$, there is an operation b of t_j such that $u = u_1bu_2$ and b left-conflicts with $u_{1|t_i}$. By Lemma 5, there must be an operation $a \in u_{1|t_i}$ such that b left-conflicts with a. That means $(t_i, t_j) \in A$.

The following schedule u of $\{t_1, t_2\}$ shows that $u \in ISER$, but $u \notin CSER$:

$$u =$$

$[BA,\ \textit{deposit},\ 100]_1[BA,\ \textit{withdraw},\ 50]_1[BA,\ \textit{deposit},\ 10]_2[BA,\ \textit{balance},\ -]_1$

In $G_I(u)$, $B = \{(t_2, t_1)\}$. But in $G_c(u)$, $A = \{(t_1, t_2),\ (t_2, t_1)\}$.

3.4. K-serializable schedules

In order to define k-serializable schedules, we first give some new notions. Let u be a schedule of set $T = \{t_1, ..., t_n\}$, T_k be a subset of T of size k ($T_k = T$ if $k \geq n$). We denote by $u_{|T_k}$ the subsequence of u including only operations of transactions in T_k. Given the conflict-serialization graph $G_c(u) = (T, A)$, we define the graph $G_{T_k}(u) = (V_k, A_k)$ as follows:

- The set of nodes $V_k = T \cup \{[T_k]\} \setminus T_k$ ($[T_k]$ is regarded as a new node);
- The arc set A_k is defined as follows: for every $(t_i, t_j) \in A$,

  if $t_i \notin T_k$ and $t_j \notin T_k$ then $(t_i, t_j) \in A_k$,

  if $t_i \notin T_k$ and $t_j \in T_k$ then $(t_i, [T_k]) \in A_k$,

  if $t_i \in T_k$ and $t_j \notin T_k$ then $([T_k], t_j) \in A_k$.

  Apart from the ones given above, A_k does not contain any other arcs.

Definition 6. A schedule u of set $T = \{t_1, ..., t_n\}$ of transactions is k-serializable if and only if there exists a subset T_k of T such that $G_{T_k(u)}$ is acyclic and $u_{|T_k}$ is serializable.

The class of k-serializable schedules is denoted by SER_k. Given a constant k, testing whether $u_{|T_k}$ is serializable takes time $O(k!)$, choosing subset T_k of T of size k takes time $O(n^k)$, therefore recognizing the schedules of SER_k can be done in polynomial time in n.

We state the following theorem.

Theorem 4. *The following statements hold:*

(1) $SER_o = SER_1 = CSER$.

(2) For each serializable schedule u there exists some $k \geq 0$ such that $u \in SER_k$.

(3) For every k, $SER_k \subset SER$.

(4) For every $k \geq 1$, $SER_k \subset SER_{k+1}$.

Proof.

(1) These follow directly from the definitions of SER_o, SER_1 and $CSER$.

(2) This assertion follows directly from the definition of k-serializability.

(3) To prove $SER_k \subseteq SER$, let us take a schedule u of transactions $\{t_1, ..., t_n\}$ in SER_k. If $k \geq n$ then u is serializable by the definition of k-serializability. Otherwise, if $k < n$, there exists a subset T_k of T of size k such that graph $G_{T_k}(u)$ is acyclic and $u_{|T_k}$ is serializable. Regarding $u_{|T_k}$ as a transaction, using the proof of Theorem 2, we have $u = u_1 u_{|T_k} u_2$, where u_1 and u_2 are serial schedules. Since relation a is transitive, schedule u is serializable, too. In both cases u is in SER.

The assertion $SER_k \subset SER$ follows from (4).

(4) We first prove $SER_k \subseteq SER_{k+1}$. Let u be a k-serializable schedule of transactions $T = \{t_1, ..., t_n\}$. If $k+1 \geq n$, then schedule u is $(k+1)$-serializable (because of the fact that u is k-serializable, it is also serializable). Otherwise, $k+1 < n$, then there exists a subset T_k of T of size k such that graph $G_{T_k}(u) = (V_k, A_k)$ is acyclic and $u_{|T_k}$ is serializable. To prove $u \in SER_{k+1}$, we must show a subset T_{k+1} of T of size $k+1$ such that $G_{T_{k+1}}(u)$ is acyclic and $u_{|T_{k+1}}$ is serializable.

We denote:

$$\begin{aligned} T_k^+ &= \{t_i \mid t_i \in V_k, \ \ ([T_k], t_i) \in A_k\} \quad \text{and} \\ T_k^- &= \{t_i \mid t_i \in V_k, \ \ (t_i, [T_k]) \in A_k\}. \end{aligned}$$

There are 3 possible cases for T_k^+ and T_k^- as follows: at least one of them is not empty, or both are empty. We next consider these cases.

The first case: $T_k^+ \neq \emptyset$. For the sake of simplicity, suppose $T_k^+ = \{t_1, ..., t_m\}$, where $m \geq 1$. We first prove that we can choose some node t_p in T_k^+ such that in graph $G_{T_k}(u)$ there are no paths of length ≥ 2 from $[T_k]$ to t_p. We suppose in contradiction that for every $t_i \in T_k^+$ in $G_{T_k}(u)$ there exists a path of length ≥ 2 from $[T_k]$ to t_i. Therefore, for every node $t_i \in T_k^+$, in $G_{T_k}(u)$ there exists a path of length ≥ 1 from some node of T_k^+ to t_i. Since $G_{T_k}(u)$ is acyclic, we can sort $\{t_1, ..., t_m\}$ in a linear order consistent with the order of $G_{T_k}(u)$ restricted on $\{t_1, ..., t_m\}$, for example $t_{i_1} t_{i_2} ... t_{i_m}$, where $(i_1 i_2 ... i_m)$ is a permutation of $(12...m)$. It contradicts to the hypothesis that for t_{i_1} in $G_{T_k}(u)$ there exists a path of length ≥ 1 from some node of T_k^+ to t_{i_1}.

With chosen node t_p, let $T_{k+1} = T_k \cup \{t_p\}$. Given $G_{T_k}(u) = (V_k, A_k)$, by the property of t_p, graph $G_{T_{k+1}}(u) = (V_{k+1}, A_{k+1})$ can be defined from $G_{t_k}(u)$ as follows.

- The set of nodes $V_{k+1} = V_k \cup \{[T_{k+1}]\} \setminus \{[T_k], t_p\}$.
- For arc set A_{k+1}:
  if $(t_i, t_j) \in A_k$, t_i and $t_j \notin \{[T_k], t_p\}$ then $(t_i, t_j) \in A_{k+1}$,
  if $(t_i, [T_k]) \in A_k$, $t_i \notin \{[T_k], t_p\}$ then $(t_i, [T_{k+1}]) \in A_{k+1}$,
  if $(t_i, t_p) \in A_k$, $t_i \notin \{[T_k], t_p\}$ then $(t_i, [T_{k+1}]) \in A_{k+1}$,
  if $([T_k], t_j) \in A_k$, $t_j \notin \{[T_k], t_p\}$ then $([T_{k+1}], t_j) \in A_{k+1}$,
  if $(t_p, t_j) \in A_k$, $t_j \notin \{[T_k], t_p\}$ then $([T_{k+1}], t_j) \in A_{k+1}$.

We first prove that $G_{T_{k+1}}(u)$ is acyclic. Suppose in contradiction that, there is a cycle in $G_{T_{k+1}}(u)$. That cycle must involve $[T_{k+1}]$, if not, it is also a cycle of $G_{T_k}(u)$ and the graph $G_{T_k}(u)$ is cyclic. Suppose that the part of the cycle which involves $[T_{k+1}]$ is $...t_i[T_{k+1}]t_j...$ (i.e. $(t_i, [T_{k+1}])$ and $([T_{k+1}], t_j)$ are arcs of A_{k+1}). We now see the above cases which define the arcs.

- If $(t_i, [T_k]) \in A_k$ and $([T_k], t_j) \in A_k$, then $G_{T_k}(u)$ is cyclic.
- If $(t_i, t_p) \in A_k$ and $(t_p, t_j) \in A_k$, then $G_{T_k}(u)$ is cyclic.
- If $(t_i, [T_k]) \in A_k$ and $(t_p, t_j) \in A_k$, then $G_{T_k}(u)$ is cyclic, because $([T_k], t_p) \in A_k$.
- If $(t_i, t_p) \in A_k$ and $([T_k], t_j) \in A_k$, then there is a path of length ≥ 2 from $[T_k]$ to t_p in $G_{T_k}(u)$, which contradicts to the choice of t_p.

All the cases lead to a conflict, so we have proved $G_{T_{k+1}}(u)$ is acyclic.

We now only need to prove that $u_{|T_{k+1}}$ is serializable. Regarding $u_{|T_k}$ as a transaction, $u_{|T_{k+1}}$ is a schedule of two transactions $\{u_{|T_k}, t_p\}$. By $t_p \in T_k^+$, in the conflict-serialization graph $G_c(u_{|T_{k+1}})$ there is only one arc $(u_{|T_k}, t_p)$. Using the proof of Theorem 2, we have $u_{|T_{k+1}} = u_{|T_k} t_p$, additionally, $u_{|T_k}$ is serializable, we conclude $u_{|T_{k+1}}$ is also serializable.

For the second case $T_k^- \neq \emptyset$, we can prove similarly that u is $(k+1)$-serializable.

For the last case, $T_k^+ = T_k^- = \emptyset$, we can choose any $t_p \notin T_k$. Let $T_{k+1} = T_k \cup \{t_p\}$, we can prove similarly that $G_{T_{k+1}}(u)$ is acyclic and $u_{|T_{k+1}}$ is serializable.

For every case we have proved that $SER_k \subseteq SER_{k+1}$. To prove $SER_k \subset$ $\subset SER_{k+1}$, using Examples 1 and 2, we consider the following schedule u of set of transactions $T = \{t_1, ..., t_{k+1}\}$:

$$u = D_1 D_2 W_1 D_3 W_2 D_4 ... W_{k-1} D_{k+1} W_k W_{k+1}.$$

For every i $(i = 1, ..., k+1)$, transaction

$$t_i = D_i W_i = [BA,\ \textit{deposit},\ a_i]\ \ [BA,\ \textit{withdraw},\ b_i],$$

where a_i, b_i are reals, $a_i > b_i$. For any state of BA, return value of every operation in u are *"ok"*, therefore, schedule u is reducible to any serial schedule of the same transactions, we have $u \in SER_{k+1}$. Since the set of arcs of the conflict-serialization graph $G_c(u)$ of u is $t_1 \leftrightarrow t_2 \leftrightarrow ... \leftrightarrow t_{k+1}$, with any choice of subset T_k of T of size k, the graph $G_{T_k}(u)$ is always cyclic. We conclude that schedule $u \notin SER_k$.

4. Conclusion

The correctness criteria in Section 3 can be used as a basis to concurrency control methods. Particularly, the conflict-serializability criterion can be applied to most common methods: two-phase locking protocol and timestamp based concurrency control. Using semantics of operations allows us to reap the benefits of decreasing conflicts between operations. However, it increases the complexity of concurrency control algorithms, because the conflict between two operations, for example $[x,\ op,\ inp]$ and $[x',\ op',\ inp']$, depends on all the parameters. In many applications, we can decrease the complexity by using concrete semantics of operations. For example, the conflict between operations on bank accounts does not depend on parameter "*inp*" (see Example 1 and 2).

The correctness criteria presented above does not depend on database states. However, a concurrent execution of transactions always departs from a consistent state of the database, hence we can think of correctness criteria as depending on states. Using correctness criteria depending on states, we hope that concurrency degree of the system can increase. For example, we see two sequences $di.wj$ and $wj.di$ of operations on bank account in Example 1: $di = [BA,\ deposit,\ I]$ and $wi = [BA,\ withdraw,\ j]$. Generally, the sequences are not equivalent, but they are equivalent for states of BA greater than j.

Clearly, the increase in concurrency obtained by the different correctness criteria is achieved at the cost of additional overhead. Whether this cost is worthwhile in practice depends on the characteristics of transactions and data, and it requests that we need to evaluate our approach experimentally in future work.

References

[1] **Agrawal D., Abbadi A.E. and Singh A.K.**, Consistency and orderability: semantics-based correctness criteria for databases, *ACM Trans. Database Syst.*, **18** (3) (1993), 460-486.

[2] **Beeri C., Bernstein P. and Goodman N.**, A model for concurrency in nested transactions, *J. ACM*, (1989).

[3] **Benczúr A. and Quang C.K.**, A model for concurrent transactions in systems using abstract data types, *Proceedings of the Fourth Symposium on Programming Languages and Software Tools, Visegrád, Hungary, June 1995.*, 231-240.

[4] **Bernstein P. and Goodman N.**, Multiversion concurrency control-theory and algorithms, *ACM Trans. Database Sys.*, **8** (1983), 465-483.

[5] **Korth H.F. and Silberschatz A.**, *Database system concepts*, 1991.

[6] **Papadimitriou C.H.**, The serializability of concurrent database updates. *J. ACM*, **26** (4) (1979), 631-653.

[7] **Ullman J.D.**, *Principles of database and knowledge-base systems*, Computer Science Press, 1988.

[8] **Vianu V. and Vossen G.**, Conceptual-level concurrency control of relational update transactions, *Proceedings of 2nd International Conference on Database Theory*, 1988, 353-367.

[9] **Vingralek R., Ye H., Breitbart Y. and Schek H.-J.**, Unified transaction model for semantically rich operations.

[10] **Weilh W.E.**, Local atomicity properties: Modular concurrency control for abstract data types, *ACM Trans. Prog. Lang. Syst.*, **11** (2) (1989), 249-283.

[11] **Weilh W.E.**, The impact of recovery on concurrency control, *Proceeding of the 8th ACM Symposium on Principles of Database Systems*, 1989, 259-269.

A. Benczúr and Chu Ky Quang
Department of General Computer Science
Eötvös Loránd University
VIII. Múzeum krt. 6-8.
H-1088 Budapest, Hungary
abenczur@palmal.elte.hu