Theoretical Computer Science 38 (1985) 1-16
North-Holland

MAXIMAL SERIALIZABILITY OF ITERATED TRANSACTIONS

M.P. FLÉ and G. ROUCAIROL
Laboratoire de Recherche en Informatique, Université de Paris-Sud, Bâtiment 490, 91405 Orsay, France

Communicated by M. Nivat
Received December 1983
Revised December 1984

Abstract. The serializability condition is usually considered in order to maintain the consistency of a Database in the presence of conflicting accesses to the Database performed by concurrent transactions. This serializability condition is considered herein as a general synchronization problem among transactions (or processes) which can be iterated infinitely often. The behaviour of such a system of transactions is represented by an infinite word over the alphabet of the operations performed by the transactions. Then a characterization of the prefixes of such behaviours satisfying the serializability condition—so-called correct behaviours—is given and it is shown that the set of all correct behaviours can be controlled by a finite automaton. As an example, the classical 'dining philosophers' problem is shown to be entirely modelled and solved as a serializability problem.

1. Introduction

Mainly studied in the framework of Data Base Systems, the serializability problem is a very general synchronization problem which can be defined as follows: let us consider a system characterized by a so-called consistency property of its states. (In an airline reservation system, such a property could be: one seat in a plane cannot be booked to more than one passenger.) Let us assume that any transaction or process operating on the system states, transforms individually a state, correct with respect to the consistency property, into another correct state. Clearly, any sequential composition of the transactions preserves the consistency property. Thus, the serializability problem is to synchronize the transactions in order to allow only concurrent behaviours which are equivalent to some sequential composition of the transactions.

In the literature on serializability [1], a transaction is considered to be a finite sequence of operations. In this paper we assume that the sequence of operations performed by a transaction can be infinitely often repeated as for instance might behave a pre-existing service process in an operating system. As an example, we shall formalize in Section 7 a resource allocation problem (the classical 'dining philosophers' problem) as a serializability problem. Therefore, a behaviour of a system of transactions will be understood herein as an infinite behaviour and we shall represent a behaviour by an infinite word formed by interleaving the transactions.

The equivalence which is generally used is such that two behaviours are said equivalent if the relative ordering of so-called conflicting operations is the same in both behaviours. (In Data Base systems, conflicts are deduced from the way operations of different transactions access to shared data.) If we call correct a behaviour which is equivalent in that sense to some sequential behaviour, we shall say (Section 3) that a system of transactions is maximally serializable (or maximally concurrent) if its set of possible behaviours is the set of correct ones. In Section 4 we characterize a maximally serializable system by a property of the prefixes of its behaviours. This property extends a result from Papadimitriou [9] and allows us to point out that the classical 'two-phase locking' synchronization algorithm [5] does not guarantee maximal concurrency (Section 5). This property will be used in Section 6 in order to show that the language of the prefixes of behaviours of a maximally serializable system is a regular language (the main result). This result is important in the sense that it shows that the serializability of iterated transactions can be finitely controlled. Moreover, some recent developments [3] point out that this result gives new insights into the theory of partially commutative free monoids.

2. Notations and basic definitions

Let X be a finite alphabet. X^* is the free monoid generated by X. λ is the empty word. X^ω is the set of infinite words over X.

If u is in X^*, u^ω is the word obtained by catenating u infinitely often with itself, $\text{proj}_Y(u)$ ($Y \subseteq X$), the erasing homomorphism which suppresses from u the symbols not in Y; let x be in $X^* \cup X^\omega$; $u \leqslant x$ (respectively $u < x$) means that u is a prefix (respectively strict prefix) of x; $|x|_a$, $a \in X$, denotes the number of occurrences of a in x; $r(x)$ denotes the set of strict right factors of x.

If W is a subset of $X^* \cup X^\omega$, $\text{Pref}(W)$ is the set of prefixes of the items of W. If x is in $X^* \cup X^\omega$, $\text{Pref}(x)$ is the set of prefixes of x.

The interval $[1, n]$, $n \in \mathbb{N}$, will be noted $[n]$.

A subset K of X^ω is said *closed* if K is exactly the set of words of X^ω every prefix of which is in $\text{Pref}(K)$, i.e., $K = \{x \in X^\omega \mid \forall u < x,\ u \in \text{Pref}(K)\}$. A subset B of X^ω is said *closed in a subset* H of X^ω if B is the intersection of a closed set and the set H.

Definition 2.1. A *transaction* T_i, $i \in \mathbb{N}$, is a finite sequence of distinct operations: $T_i = a_{i,1} \ldots a_{i,m_i}$; we shall call A_i the *set of operations of a transaction* T_i.

In the sequel, we will consider concurrent computations of several transactions, each being possibly infinitely often iterated.

Let $T = \{T_1, T_2, \ldots, T_n\}$ be a family of n transactions.

Definition 2.2. A *concurrent computation* x of transactions in T is an infinite word over $A = \bigcup_{i=1}^{n} A_i$ obtained by shuffling possibly infinite loops of the transactions; we shall note by $\mathrm{CC}(T)$ the set of concurrent computations of transactions in T, i.e.,

$$\mathrm{CC}(T) = \{x \in A^{\omega} \mid \forall i \in [n], \operatorname{proj}_{A_i}(x) \in \{T_i\}^* \cup \{T_i^{\omega}\}\}.$$

In the sequel, the term computation will be understood as concurrent computation. Now, we are able to give the following definition.

Definition 2.3. A *transaction system* is a triple $\mathrm{TS} = \langle T, B, R\rangle$, where:
- $T = \{T_1, \ldots, T_n\}$ is a finite set of transactions. In the sequel, we shall assume that the sets of operations of different transactions are disjoint, i.e.,

$$\forall i, j \in [n], \quad A_i \cap A_j \neq \emptyset \Leftrightarrow i = j.$$

- B is the set of behaviours of TS; it is a subset of the set $\mathrm{CC}(T)$.
- $R \subseteq A \times A$ is a symmetric relation, the so-called 'conflict relation' among the operations of different transactions, i.e.,

$$\forall i \in [n], \quad R \cap A_i \times A_i = \emptyset.$$

Example 2.4. Let us consider two transactions T_1 and T_2 performing the following sequences of instructions on two variables A and B satisfying the consistency predicate '$A = B$':

$$T_1 : A := A * 2;\ B := B * 2 \qquad T_2 : A := A + 10;\ B := B + 10$$

Calling a (respectively a') the first operation of T_1 (respectively T_2) and b (respectively b') the second operation of T_1 (respectively T_2), a transaction system TS modelling the concurrent and infinite behaviours of these transactions can be given by

$$T = \{T_1, T_2\},$$

$$B = \{ab, aa'bb', a'b', a'ab'b\}^{\omega},$$

$$R = \{(a, a'), (a', a), (b, b'), (b', b)\}.$$

We can remark that this transaction system has the property that all its behaviours preserve the consistency predicate.

Remarks. We consider each operation performed by a transaction as atomic. This assumption is not too restrictive as far as we are concerned in this paper by the relative ordering of conflicting operations only.

Parallelism among transactions is represented in behaviours by the fact that some occurrences of transactions can start while some others are not achieved. For instance, in $(aa'bb')^{\omega}$ (see Example 2.4), T_2 begins before T_1 is achieved.

3. Correct computations and maximal serializability of a transaction system

Definition 3.1. A computation of a transaction system is *sequential* if it is obtained by some catenation of the transactions only, i.e., if it is an item of T^ω.

A computation of a transaction system is said *correct* if it is equivalent to some sequential computation. The equivalence which is mainly used in the literature on serializability can be formalized by an equivalence defined by Keller [8]. It concerns the comparison of occurrences of conflicting operations.

Definition 3.2. Let x and y be two computations. x is said *equivalent* to y $(x \sim y)$ if and only if:

(i) $\forall a \in A$, $\text{proj}_{\{a\}}(x) = \text{proj}_{\{a\}}(y)$ (identical occurrences of operations),

(ii) $\forall (a, b) \in R$, $\text{proj}_{\{a,b\}}(x) = \text{proj}_{\{a,b\}}(y)$ (identical ordering of occurrence of conflicting operations).

Example 3.3. The following computations taken from Example 2.4 are equivalent:

$$x = (aa'bab'b)^\omega, \qquad y = (aba'b'ab)^\omega.$$

Definition 3.4. A computation x is *correct* if and only if there exists a sequential computation equivalent to x, i.e., $\exists y \in T^\omega$ such that $x \sim y$.

We denote by Cor(TS) the set of correct computations of a transaction system TS.

Example 3.5. The computation x taken from Example 3.3 is correct.

A semantic justification of this notion of correctness can be given by the following arguments: Let us assume that the conflict relation is derived from the way two operations access to a common variable (read-write and write-write conflict); then we know (Keller [8]) that the history of values assigned to a variable is the same for two equivalent behaviours under any interpretation; assuming that every transaction preserves some consistency predicate (which is a property of the variables used by the operations), then so does any sequential behaviour and therefore so does any correct behaviour.

Definition 3.6. A transaction system TS is said *maximally serializable* if and only if its given set of behaviours B is exactly the set of correct computations: $B = \text{Cor(TS)}$.

This notion of maximal serializability can be understood as a notion of maximal concurrency. As a matter of fact, concurrency is represented in our formalism by the possibility of shuffling sequences of operations. Therefore, the more behaviours has a transaction system, the more possibilities of shuffling the transactions sequences exist, so the more concurrent is this transaction system.

4. Characterization of maximal serializability

In order to characterize the set of correct computations, we associate to each prefix of a computation a precedence relation among occurrences of transactions. This relation is deduced from the order of occurrences of conflicting operations.

Definition 4.1. Let u be a prefix of a computation. We say that the hth occurrence of T_i *precedes* in u the kth occurrence of T_j and we write $(T_i, h) <_u (T_j, k)$ if and only if:

- either $i = j$, $h < k$, and the first operation of T_i occurs at least $k-1$ times in u, i.e., $|u|_{a_{i,1}} \geq k-1$.
- or $i \neq j$ and there exists one operation a in A_i, conflicting with one operation b in A_j, such that the hth occurrence of a in u must precede the kth occurrence of b if it occurs in u or in any extension of u, i.e., $\exists (a, b) \in R \cap A_i \times A_j$, $(v_1, v_2, v_3) \in A^* \times A^* \times A^*$, such that

$$u \in \{v_1 a v_2,\ v_1 a v_2 b v_3\}, \quad |v_1|_a = h-1, \quad |v_1 a v_2|_b = k-1.$$

Example 4.2. Let us consider the transaction system given in Example 2.4: $u = a$ implies $(T_1, 1) <_u (T_1, 2)$ and $(T_1, 1) <_u (T_2, 1)$ because $(a, a') \in R$. $u = aa'$ implies $(T_1, 1) <_u (T_1, 2)$, $(T_1, 1) <_u (T_2, 1)$, $(T_2, 1) <_u (T_2, 2)$ and $(T_2, 1) <_u (T_1, 2)$ because $(a', a) \in R$.

For $u = aa'b'$, we have these four relations and $(T_2, 1) <_u (T_1, 1)$ because $(b', b) \in R$.

Let us denote $<_u^*$ the transitive closure of $<_u$ and let us call Ord(TS) the set of prefixes u of computations for which the relation $<_u^*$ is an order relation:

$$\mathrm{Ord(TS)} = \{u \in \mathrm{Pref(CC}(T)) \mid <_u^* \text{ is an order relation}\}.$$

For instance, in Example 4.2, aa' is in Ord(TS) and $aa'b'$ is not.

In the following definition, we extend the $<_u$ relation from prefixes to computations.

Definition 4.3. Let x be a computation. The relation $<_x$ is the union of the relations $<_u$ for every prefix u of x, i.e., $<_x = \bigcup_{u<x} <_u$.

This relation allows to give a necessary and sufficient condition for a prefix of a computation to be extended into a correct computation and to characterize correct computations.

Proposition 4.4.1. Ord(TS) = Pref(Cor(TS)).

Proposition 4.4.2. *Let x be in A^ω.*

The following conditions are equivalent:
(i) *x is correct,*
(ii) *x is a computation and every prefix of x is in* Ord(TS),
(iii) *x is a computation and $<_x^*$ is an order relation.*

In order to prove Propositions 4.4.1 and 4.4.2, we introduce some notations.

Notations. We know that each T_i can be written as $a_{i,1} \dots a_{i,m_i}$. We shall denote by $|u|_{T_i}$ the number $|u|_{a_{i,1}}$ of transactions started in a prefix u of a computation, by $A(a)$ the alphabet containing an operation a, and by $T(a)$ the transaction whose alphabet is $A(a)$. This is well defined since $\forall i, j \in [n]$, $i \neq j \Leftrightarrow A_i \cap A_j = \emptyset$.

The proof of Propositions 4.4.1 and 4.4.2 is based upon the following lemmas.

Lemma 4.5. $u \in \text{Ord(TS)} \Rightarrow \exists z \in A^*$ *such that $<_{uz}^*$ is an order relation and $\forall i \in [n]$,* $\text{proj}_{A_i}(uz) = (T_i)^k$ *with* $k = |u|_{T_i}$.

This means that if u is in Ord(TS), there always exists a way of ending all the transactions yet started in u, while staying in Ord(TS).

Proof of Lemma 4.5. We consider a total order over $\{T_1, \dots, T_n\}$ which contains the relation

$$L = \{(T_i, T_j) : (T_i, |u|_{T_i}) <_u^* (T_j, |u|_{T_j}),\ i, j \in [n]\},$$

induced by the $<_u^*$ relation over the last occurrences of the transactions.

Since $<_u^*$ is an order relation, so is L; then we can build by induction the sequences $(k_i)_{i \in [n]}$, $(w_i)_{i \in [n]}$ such that T_{k_i} is one minimal element, for the relation L, of the set $\{T_1, \dots, T_n\} - \{T_{k_1}, \dots, T_{k_{i-1}}\}$.

$w_i \in r(T_{k_i})$ and $\text{proj}_{A_{k_i}}(u) w_i = (T_{k_i})^s$ where $s = |u|_{T_i}$.

Let $z = w_1 \dots w_n$; obviously, $<_{uz}^*$ is an order relation. □

Lemma 4.6. *Let x and y be two computations. If $x \sim y$, then $<_x^* = <_y^*$.*

This is a direct consequence of the definition of $\sim$ and $<_x^*$.

Lemma 4.7. *Let x and y be two computations with identical occurrences of operations, i.e., $\forall a \in A$,* $\text{proj}_{\{a\}}(x) = \text{proj}_{\{a\}}(y)$. *If $<_x^*$ and $<_y^*$ are identical and order relations, then $x \sim y$.*

Proof. We obtain the claim of the lemma by assuming $\text{proj}_{\{a,b\}}(x) \neq \text{proj}_{\{a,b\}}(y)$ where $(a, b) \in R \cap A_i \times A_j$ for some i and j in $[n]$. Then $ua < x$ (1), and $vb < y$ (2) for some u and v such that $|u|_a = |v|_a$ and $|u|_b = |v|_b$ (3).

(1) implies $(T_i, |u|_a+1) <_x (T_j, |u|_b+1)$ and then $(T_i, |u|_a+1) <_y^* (T_j, |u|_b+1)$, (2) and (3) imply $(T_j, |u|_b+1) <_y (T_i, |u|_a+1)$.

This contradicts the fact that $<_y^*$ is an order relation. $\square$

Lemma 4.8. *If x is a sequential computation, then $<_x^*$ is an order relation.*

This follows straightforwardly from the definition of $<_x^*$.

Lemma 4.9. *Let x be a computation. $<_x^*$ is an order relation $\Leftrightarrow$ $\forall u < x$, $<_u^*$ is an order relation.*

Proof. The proof comes from the fact that for all prefixes u and v of some computation x, we have:

$$u \leqslant v \Rightarrow <_u \subseteq <_v. \qquad \square$$

Lemma 4.10. *Let x be a computation. $<_x^*$ is an order relation $\Rightarrow x$ is correct.*

Proof. Let $x \in \mathrm{CC}(T)$. We will build a sequential computation y such that $<_x^* = <_y^*$ and $\forall a \in A$, $\mathrm{proj}_{\{a\}}(x) = \mathrm{proj}_{\{a\}}(y)$. Due to Lemma 4.7, we then have $x \sim y$ and so x will be proved to be correct.

Let $X = \{(T_i, k) : k \leqslant |x|_{T_i}, i \in [n]\}$.

Let us prove the two following properties of X.

Property 1. $\forall (T_i, k) \in X$, *the set* $\{t \mid t \in X, t <_x (T_i, k)\}$ *is finite.*

This means that, in the set X, the number of immediate predecessors of an occurrence of a transaction is finite.

This is due to the fact that, since $(T_i, k) \in X$, every operation of this transaction occurs at least k times in x (see the definitions of a transaction and of $\mathrm{CC}(T)$), so it cannot exist an infinity of occurrences of operations before the kth occurrence of a finite number of operations.

Property 2. $\forall t \in X$, *there does not exist any infinite sequence* $(T_{i_j}, h_j)_{j \in \mathbb{N}}$ *such that*

$$\forall j \in \mathbb{N}, \quad (T_{i_{j+1}}, h_{j+1}) <_x (T_{i_j}, h_j) \quad \text{and} \quad (T_{i_1}, h_1) <_x t.$$

This means that no element of X has an infinite chain of predecessors.

Proof of Property 2. Suppose such an infinite chain exists; then it contains an infinity of couples $(S, k_j)_{j \in \mathbb{N}}$ (since the number of transactions is finite). Then we would have an infinite sequence of strictly decreasing integers (since $(S, k_j) <_x^* (S, k_i) \Rightarrow k_j < k_i$). This is impossible.

By Properties 1 and 2 and Koenig's lemma, each item of X has a finite number of predecessors in the relation $<_x^*$ and we can define the following mapping:

$l: X \to \mathbb{N}$, where

$$\forall t \in X, \quad l(t) = \begin{cases} 0 & \text{if } t \text{ is without any predecessor,} \\ \sup\{j: \exists t_1, \ldots, t_j \text{ and } t_1 <_x t_2 \ldots t_j <_x t\}. \end{cases}$$

$l(t)$ is the length of the longest chain of predecessors of t.

Let $X_k = \{t \in X \mid l(t) = k\}$. For every k, X_k is finite since $X_k \subseteq \{(T_i, j),\ j \leqslant k+1,\ i \in [n]\}$.

Let w_k be the word formed by an arbitrary ordering of the items of X_k and let $z = w_1 \ldots w_k \ldots$. Let y be the word over the alphabet of the transactions obtained by erasing from z the numbers of occurrence of each transaction, i.e., $\forall i \in \mathbb{N}$, if $z[i] = (T_k, j)$, then $y[i] = T_k$.

In other words: $\forall i \in \mathbb{N}$, $\forall k \in [n]$, $z[i] = (T_k, j) \Leftrightarrow y[i]$ is the jth occurrence of T_k in y. Then it is easy to see by construction that

(1) $<_x = <_y$.

(2) $\forall a \in A$, $\text{proj}_{\{a\}}(x) = \text{proj}_{\{a\}}(y)$. □

Proof of Proposition 4.4.1. Every u in Ord(TS) can be extended into a word uz where z is defined as in Lemma 4.5.

$<_{uz}^*$ being an order relation, so is $<_{uzS\omega}^*$ for some transaction S; then, from Lemma 4.10, uzS^ω is correct.

Conversely, from Lemmas 4.6 and 4.8, every u in Pref(Cor(TS)) is in Ord(TS). □

The proof of Proposition 4.4.2 is obviously obtained from Lemmas 4.6, 4.8, 4.9, and 4.10.

From Propositions 4.4.1 and 4.4.2 we deduce the following characterization theorem for maximal serializability, which generalizes a result from Papadimitriou [9].

Theorem 4.11. *$\langle T, B, R\rangle$ is maximally serializable* ($B = \text{Cor(TS)}$) *if and only if B is closed in* CC(T) *and* Pref(B) = Ord(TS).

Proof. *Necessary condition*: The fact that Pref(B) = Ord(TS) is a direct consequence of Proposition 4.4.1.

Let $K = \{x \in A^\omega \mid \forall u < x, u \in \text{Ord(TS)}\}$. Clearly K is a closed set and, from Proposition 4.4.2, $B = K \cap \text{CC}(T)$; then B is closed in CC(T).

Sufficient condition: Suppose x is correct; then by Proposition 4.4.2, every prefix u of x is in Ord(TS), so, u is in Pref(B). Since B is closed in CC(T), $B = K \cap \text{CC}(T)$ where K is a closed set; then x is in K; on the other hand, x being correct, x is a computation. So x is in B.

Suppose x is in B. Then x is a computation and every prefix of x is in Ord(TS), so, by Proposition 4.4.2, x is correct. □

5. Comparison with two-phase locking algorithm

The 'two-phase locking' [5] algorithm is used in Data Base Systems in order to control transactions which access to shared entities. It consists in interleaving lock and unlock operations in the transactions in the following way:

For every transaction S,

- if an operation a of S accesses to an entity e, then a lock(e) must precede a without any unlock(e) between it and a.
- if an operation $a = \text{lock}(e)$ has been interleaved in S, then there is no other lock(e) before it in S.
- there is an unlock operation c in S such that neither another unlock operation precedes c nor any lock operation occurs after c.

A schedule is any shuffle of the interleaved transactions. Such a word is said legal if two lock(e)'s cannot occur in it without any unlock(e) between them. We shall say that a computation is legal if it is the projection, over the alphabet of the operations of the transactions, of a legal schedule.

In order to show that the 'two-phase locking' algorithm does not guarantee maximal concurrency, we are going to give an example of a transaction system where, for any interleaving of the lock and unlock operations corresponding to the 'two-phase locking' algorithm, there exists a correct computation which is not a legal one. (One can verify that the notion of correctness of a computation, when the conflict relation is deduced from the way two operations access to a common variable, is the same as the notion of consistent schedule in [5].)

Let us consider the following transaction system:

$$T = \{T_1, T_2, T_3\} \quad \text{with } T_1 = a(e),\ T_2 = b(e);\ c(f),\ T_3 = d(f).$$

$$R = \{(a(e), b(e)), (b(e), a(e)), (c(f), d(f)), (d(f), c(f))\}.$$

The interleaving for T_1 and T_3 is unique:

$$T_1 : \text{lock}(e);\ a(e);\ \text{unlock}(e),$$

$$T_3 : \text{lock}(f);\ d(f);\ \text{unlock}(f).$$

For T_2, lock(f) must precede unlock(e). Then, one can verify that the word $b(e)\,;a(e)\,;d(f)$ is a prefix of a correct computation but is not a prefix of a legal one.

6. Regularity of maximal serializability

In this section we show (Theorem 6.1) that the set of prefixes of correct computations of a transaction system is a regular language. The rule of acceptance of correct computations is then characterized over the automaton which recognizes the set of their prefixes.

Since, for a transaction system TS, Pref(Cor(TS)) = Ord(TS) (Proposition 4.4.1), we prove the following.

Theorem 6.1. Ord(TS) *is a regular language.*

Proof. The regularity of Ord(TS) is obtained by proving that the number of equivalence classes of the relation $\hat{}$ defined by

$$\forall u, v \in \mathrm{Ord(TS)},$$

$$u \,\hat{}\, v \Leftrightarrow \forall z \in A^*, (uz \in \mathrm{Ord(TS)} \Leftrightarrow vz \in \mathrm{Ord(TS)}),$$

is finite.

For every u in Ord(TS), we consider the set $C(u)$ of some extensions of u compatible with the $<_u^*$ relation. These extensions allow to end every transaction started but not finished in u and also restart every transaction once more, i.e., we can state the following definition.

Definition 6.2. $C: \mathrm{Ord(TS)} \rightarrow 2^{A^*}$ is a mapping defined by:

$$\forall u \in \mathrm{Ord(TS)},$$

$$C(u) = \{z \in A^* \mid \forall i \in [n], \mathrm{proj}_{A_i}(z) \leq s_i T_i, s_i \in r(T_i), uz \in \mathrm{Ord(TS)}\}.$$

Let us consider

$$P = \{z \in A^* \mid \forall i \in [n], \mathrm{proj}_{A_i}(z) \leq s_i T_i, s_i \in r(T_i)\}.$$

P is a finite set and obviously we have: $\forall u \in \mathrm{Ord(TS)}$, $C(u) \subseteq P$. So, the number of equivalence classes of the relation '$C(u) = C(v)$' is finite. Then the regularity of Ord(TS) will come from the following proposition.

Proposition 6.3

$$\forall u, v \in \mathrm{Ord(TS)}, \quad C(u) = C(v) \Leftrightarrow u \,\hat{}\, v.$$

Proof of Proposition 6.3. *Sufficient* condition is obvious.

The *necessary* condition consists, under the assumption $C(u) = C(v)$, in proving that the following properties hold.

Property A. $\forall z \in A^*$, $uz \in \mathrm{Pref(CC(}T))$ $\Rightarrow$ $vz \in \mathrm{Pref(CC(}T))$.

Property B. $\forall z \in A^*$, *if* $uz \in \mathrm{Pref(CC(}T))$ *and* $<_{uz}^*$ *is an order relation, then* $<_{vz}^*$ *is an order relation.*

The first property is easily obtained from the definition of CC(T) and the following lemma.

Lemma 6.4. *If $C(u) = C(v)$, then,*

$$\forall i \in [n], \exists s_i \in \mathrm{Pref}(T_i),$$
$$\mathrm{proj}_{A_i}(u) = (T_i)^k s_i \textit{ for some } k \Leftrightarrow \mathrm{proj}_{A_i}(v) = (T_i)^h s_i \textit{ for some } h.$$

This means that all the operations which appear in the last occurrence of a transaction in u also appear as the last occurrence of this transaction in v and conversely. This is a direct consequence of Lemma 4.5, the definition of C and the fact that $C(u) = C(v)$.

The difficulty lies in the proof of Property B: we suppose that $<^*_{uz}$ is an order relation and we assume that there exists a circuit in the graph of the relation $<_{vz}$, i.e., there exists a p in $\mathbb{N}$ and sequences of integers $(i_j)_{j\in[p]}$ and $(k_j)_{j\in[p]}$ such that the graph $G_p^{i,k} = \{(T_{i_{j-1}}, k_{j-1}), (T_{i_j}, k_j), j \in [p]\}$ is a circuit and $G_p^{i,k} \subseteq <_{vz}$. Then we prove that it is possible to choose $G_p^{i,k}$ in order to build a word $F(z)$ such that:

(1) $F(z) \in C(u)$, and

(2) the relation $<_{vF(z)}$ contains a circuit.

Since $C(u) = C(v)$, the contradiction will come from the fact that $C(v)$ contains $F(z)$ and the relation $<^*_{vF(z)}$ is not an order relation.

The proof of Property B consists now in the construction of $F(z)$ and in proving (1) and (2).

We need the following lemma.

Lemma 6.5. *There exists a circuit $G_q^{j,h} \subseteq <_{vz}$ such that*

$$\forall r, m \in [1, q-1], \quad T_{j_r} \neq T_{j_m}. \qquad (*)$$

This means that there exists a circuit made of occurrences of different transactions.

Proof of Lemma 6.5. Suppose $(*)$ is not true in $G_p^{i,k}$, (i.e., for $q = p$, $h = k$, and $j = i$). Let r and m be such that $(r, m) \in [1, p-1]$, $T_{i_r} = T_{i_m}$ and $k_r \leqslant k_m$.

Since $(T_{i_m}, k_m) <_{vz} (T_{i_{m+1}}, k_{m+1})$, we have:

- either $i_m = i_{m+1}$ and then $(T_{i_r}, k_r) <_{vz} (T_{i_{m+1}}, k_{m+1})$,
- or $vz \in \{w_1 a w_2, w_1 a w_2 b w_3\}$ for some $(w_1, w_2, w_3) \in A^* \times A^* \times A^*$ such that $|w_1|_a = k_m - 1$, $|w_1 a w_2|_b = k_{m+1} - 1$, and $(a, b) \in R \cap A_{i_m} \times A_{i_{m+1}}$.

Since $k_r \leqslant k_m$, $|wa|_a = k_r$ for some $w \leqslant w_1$ and therefore $(T_{i_r}, k_r) <_{vz} (T_{i_{m+1}}, k_{m+1})$. Then there exists a circuit included in $G_p^{i,k}$ which does not contain the couple (T_{i_m}, k_m). By repeating the same argument we can build a circuit $G_q^{j,h}$ satisfying $(*)$. $\square$

Now let us assume that $G_p^{i,k}$ satisfies $(*)$. Then, we are able to build $F(z)$ by erasing from z all the occurrences of operations except those belonging to the single occurrence of transactions involved in $G_p^{i,k}$ or started but not finished in v, i.e., we can state the following definition.

Definition 6.6. Let us write $z = z_1 \dots z_r$, where $r \in \mathbb{N}$ and $\forall j \in [r]$, $z_j \in A$. Then $F(z) = f(z, 1) \dots f(z, r)$, where $\forall j \in [r]$,

$$f(z,j) = \begin{cases} z_j & \text{if } \exists h \in [p],\ z_j \in A_{i_h},\ |vz_1 \dots z_j|_{z_j} = k_h \text{ or } |vz_1 \dots z_j|_{z_j} = |v|_{T(z_j)}, \\ \lambda & \text{otherwise.} \end{cases}$$

The following lemmas state some useful properties of the previous erasing homomorphism F.

Lemma 6.7

$$\forall s, t \in \mathbb{N},\ s < t,\ \forall i \in \mathbb{N},\ \forall a, b \in A_i$$

$$|vz_1 \dots z_s|_a = |vz_1 \dots z_t|_b \Rightarrow |vf(z, 1) \dots f(z, s)|_a = |vf(z, 1) \dots f(z, t)|_b.$$

This means that all the operations belonging to the same occurrence of some transaction in vz are either globally suppressed or globally conserved by the homomorphism F.

The proof of this lemma can easily be obtained from the definition of F, by induction on $|vz_1 \dots z_s|_a$.

Lemma 6.8. $\forall s, t \in \mathbb{N}$, $s < t$, $\forall h \in [p]$, $\forall a, b \in A_{i_h}$, $\forall w \in A^*$:

(i) *If* $|uf(z, 1) \dots f(z, s-1)|_a = |uf(z, 1) \dots f(z, t-1)|_b$ *and* $f(z, s) = a$, *then* $|uz_1 \dots z_{s-1}|_a = |uz_1 \dots z_{t-1}|_b$.

(ii) *If* $wa \leqslant u$ *and* $|w|_a = |uf(z, 1) \dots f(z, t-1)|_b$, *then* $|w|_a = |uz_1 \dots z_{t-1}|_b$.

Lemma 6.8 is a kind of reciprocal of Lemma 6.7. We prove this by applying Lemma 6.4 and by using the definition of F.

As we said above, we now need to prove the following lemmas.

Lemma 6.9. $F(z) \in C(u)$.

Lemma 6.10. *The relation* $<_{vF(z)}$ *contains a circuit.*

Proof of Lemma 6.9. We must show that:

(1) $\forall i \in [n]$, $\text{proj}_{A_i} F(z) \leqslant s_i T_i$, $s_i \in r(T_i)$ and $uF(z) \in \text{Pref}(\text{CC}(T))$.

(2) The relation $<^*_{uF(z)}$ is an order relation.

Proof of (1): The erasing homomorphism obviously preserves the order of the operations in the transactions. Moreover, from the definition of F and Lemma 6.5 we have:

$$\forall i \in [n], \forall t \in [2, m_i], \quad |F(z)|_{a_{i,t}} \leqslant 2 \text{ and } |F(z)|_{a_{i,1}} \leqslant 1.$$

We have now obtained (1).

Proof of (2): We assume that $<^*_{uF(z)}$ is not an order relation. Then $\exists q \in \mathbb{N}$, $(h_i)_{i\in[q]}$, $h_i \in \mathbb{N}$, $(T_{j_i})_{i\in[q]}$ such that

$$\forall i \in [q-1], \quad (T_{j_i}, h_i) <_{uF(z)} (T_{j_{i+1}}, h_{i+1}) \text{ and } (T_{j_q}, k_q) = (T_{j_1}, k_1).$$

Then $\forall i \in [q-1]$, $\exists (w_1, w_2, w_3) \in A^* \times A^* \times A^*$, $(a, b) \in A_{j_i} \times A_{j_{i+1}} \cap R$ such that

$$uF(z) \in \{w_1aw_2, w_1aw_2bw_3\}, \quad |w_1| = h_i - 1, \quad |w_1aw_2|_b = h_{i+1} - 1. \qquad (\dagger)$$

$\forall i \in [q-1]$, we define:

$$g(h_i) = \begin{cases} \text{If } \exists a \in A_{j_i},\ s \in \mathbb{N},\ f(z,s) = a,\ |uf(z,1)\dots f(z,s)|_a = h_i, \text{ then} \\ \quad |uz_1 \dots z_s|_a \text{ (this is well defined due to Lemma 6.8).} \\ \text{If } \exists w \in A^*,\ a \in A_{j_i},\ wa \leqslant u,\ |wa|_a = h_i, \text{ then } h_i. \end{cases}$$

Due to $(\dagger)$, $g(h_i)$ is defined for every i in $[q-1]$.

We show that $\forall i \in [q-1]$, $(T_{j_i}, g(h_i)) <_{uz} (T_{j_{i+1}}, g(h_{i+1}))$. This obviously shows that $<_{uz}$ contains a circuit. □

Proof of Lemma 6.10. The existence of $G^{i,k}_p$ means that $\forall h \in [p-1]$, $\exists (w_1, w_2, w_3) \in A^* \times A^* \times A^*$, $(a, b) \in A_{i_h} \times A_{i_{h+1}} \cap R$ such that

$$vz \in \{w_1aw_2, w_1aw_2bw_3\}, \quad |w_1|_a = k_h - 1, \quad |w_1aw_2|_b = k_{h+1} - 1. \qquad (\dagger\dagger)$$

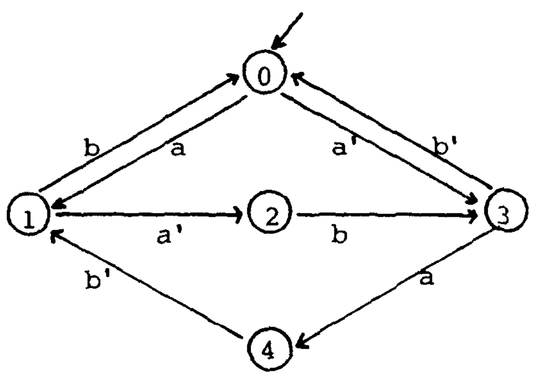
This allows us to define, $\forall h \in [p-1]$,

$$g(k_h) = \begin{cases} \text{If } \exists a \in A_{i_h},\ s \in \mathbb{N},\ z_s = a,\ |vz_1 \dots z_s|_a = k_h, \text{ then} \\ \quad |vf(z,1)\dots f(z,s)|_a \text{ (this is well defined due to Lemma 6.7)} \\ \text{If } \exists w \in A^*,\ a \in A_{i_h},\ wa \leqslant v,\ |wa|_a = k_h, \text{ then } k_h \end{cases}$$

Due to $(\dagger\dagger)$, $g(k_h)$ is defined for every h in $[p-1]$. For every $h \in [p-1]$, by looking at $(\dagger\dagger)$, we examine the different cases $v \leqslant w_1a$, $w_1a < v < w_1aw_2bw_3$, $w_1aw_2bw_3 \leqslant v$, and inside each case, the cases $vz = w_1aw_2$ and $vz = w_1aw_2bw_3$ in order to show that $(T_{k_h}, g(k_n)) <_{vF(z)} (T_{k_{h+1}}, g(k_{h+1}))$. □

This achieves the proof of Theorem 6.1. □

Example 6.11. For the transaction system of Example 2.4, the finite automaton accepting Ord(TS) is:



Remark. One might think that in order to build the set $C(u)$, only the end of every transaction yet started in u has to be considered, i.e.,

$$C(u)=\{z\in A^*, \forall i\in[n], \operatorname{proj}_{A_i}(u)\leqslant s_i, s_i\in r(T_i), uz\in \operatorname{Ord}(\mathrm{TS})\}.$$

The following counter-example shows that it is not possible.

Example 6.12. Let TS be a transaction system such that

$$T=\{a_1b_1, a_2, a_3b_3\},$$

and

$$R=\{(a_1,a_2),(a_2,a_1),(a_2,a_3),(a_3,a_2),(b_1,b_3),(b_3,b_1)\}.$$

Then, $C(a_1)=b_1$ and $C(a_1a_2)=b_1$; but $a_1a_3b_3$ is in Ord(TS) and $a_1a_2a_3b_3$ is not.

Let $\mathscr{A}=\langle q_0, Q, \tau, F\rangle$ be the automaton over the alphabet A defined by $Q=\{C(u), u\in \operatorname{Ord}(\mathrm{TS})\}$ (see Definition 6.2) is the set of states of $\mathscr{A}$, $q_0=C(\lambda)$ is the initial state, $\tau: Q\times A\to Q$ is the transition function defined by: $\forall q\in Q$, $\tau(q,a)=C(ua)$ for some u such that $C(u)=q$ (this is well defined due to Proposition 6.3), $F=Q$ is the set of terminal states.

Due to Proposition 6.3, Ord(TS) is the language accepted by $\mathscr{A}$.

In order to characterize Cor(TS) with respect to $\mathscr{A}$, we introduce the following definition (in a way similar to Buchi's [2]).

Let x be in A^ω; x passes infinitely often through a subset K of states if the following property is satisfied:

$$\forall u<x \ \exists v\in A^*, \quad u\leqslant v<x \quad \text{and} \quad \tau(q_0,v)\in K.$$

Let i be in $[n]$; let K_i be the set of states which correspond to the fact that transaction T_i is terminated, i.e.,

$$K_i=\{q\in Q\,|\,\exists u\in \operatorname{Ord}(\mathrm{TS}), \operatorname{proj}_{A_i}(u)\in\{T_i\}^* \text{ and } q=C(u)\}.$$

The following proposition says that Cor(TS) is characterized by the set of infinite words which, for every i in $[n]$, pass infinitely often through K_i.

Proposition 6.13

$$\operatorname{Cor}(\mathrm{TS})=\{x\in A^\omega\,|\,\forall u<x, \forall i\in[n], \exists v\in A^*, u\leqslant v<x, \tau(q_0,v)\in K_i\}.$$

Proof. Let x be in Cor(TS); due to Proposition 4.4.2, we have the following property:

$$\forall u<x, \quad \forall i\in[n], \quad \exists v\in \operatorname{Ord}(\mathrm{TS}), \quad u\leqslant v<x, \operatorname{proj}_{A_i}(v)\in\{T_i\}^*.$$

Then $\tau(q_0,v)\in K_i$.

Let x be an infinite word such that

$$\forall u<x, \quad \forall i\in[n], \quad \exists v\in \operatorname{Ord}(\mathrm{TS}), \quad u\leqslant v<x, \tau(q_0,v)\in K_i.$$

So $C(v) \in K_i$ and therefore $C(v) = C(w)$ for some w in Ord(TS) such that $\text{proj}_{A_i}(w) \in \{T_i\}^*$. By Lemma 6.4, this implies $\text{proj}_{A_i}(v) \in \{T_i\}^*$ and so x is a computation and, by Proposition 4.4.2, x is correct. $\square$

Let us remark that the initial state plays a particular role since we have

$$\bigcap_{i \in [n]} K_i = \{q_0\}.$$

Then, infinite words which pass infinitely often through $\{q_0\}$ are correct computations such that after some finite delay every transaction is terminated. Moreover, for such computations, it is easy to check that every prefix, for which all the transactions are terminated, is equivalent to some finite sequential computation and vice-versa. In Example 6.11, considering only those computations insures the predicate '$A = B$' become true infinitely often.

7. Application to the 'dining philosophers problem' [4]

Let us assume that the individual behaviour of one philosopher P_i is: think_i; tf_i^i; tf_i^{i+1}; eat_i; rf_i^i; rf_i^{i+1}, where, for $k \in \{i, i+1\}$, tf_i^k (respectively rf_i^k) means that philosopher P_i takes (respectively releases) fork k.

Let us consider the following transaction system: $T = \{P_1, \ldots, P_n\}$,

$$R = \{(\text{tf}_{i-1}^i, \text{rf}_i^i), (\text{rf}_i^i, \text{tf}_{i-1}^i), (\text{rf}_{i-1}^i, \text{tf}_i^i), (\text{tf}_i^i, \text{rf}_{i-1}^i), i \in [2, n]\}.$$

Let us show that in a correct computation of this transaction system, two neighbours cannot eat concurrently and conversely. Then, by applying the previous results, the maximal concurrency of the philosophers can be controlled, in a correct way, by a finite automaton.

Notation. For two operations a and b, two integers h and k and a computation x, we shall denote by $(a, h) \rightarrow_x (b, k)$ (respectively $(a, h) \Rightarrow_x (b, k)$) the fact that x can be written *uavbw* (respectively *uavbw* or *uav*), with $|u|_a = h$ and $|uav|_b = k$. Obviously, $\rightarrow_x$ and $\Rightarrow_x$ are order relations.

The fact that, in a computation x, two neighbours cannot eat concurrently is expressed by the following property Q:

Q: If $(\text{tf}_h^i, p) \rightarrow_x (\text{tf}_k^i, q)$ with $h = i-1$ and $k = i$ or $h = i$ and $k = i-1$, then $(\text{tf}_h^i, p) \rightarrow_x (\text{rf}_h^i, p) \rightarrow_x (\text{tf}_k^i, q)$.

This means that if two neighbours successively take the same fork then, between these actions, the first of them must have released this fork.

Let x be a correct computation and let us assume:

(1) $(\text{tf}_h^i, p) \rightarrow_x (\text{tf}_k^i, q)$.

From the definitions of P_i and CC(T), we have

(2) $(\mathrm{tf}^i_h, p) \rightarrow_x (\mathrm{rf}^i_h, p)$, and

(3) $(\mathrm{tf}^i_k, q) \rightarrow_x (\mathrm{rf}^i_k, q)$.

So, from (1) and (3), we have $(P_h, p) <_x (P_k, q)$. From (1) and (3), we can suppose that $(\mathrm{tf}^i_k, q) \rightarrow_x (\mathrm{rf}^i_h, p)$. This implies: $(P_k, q) <_x (P_h, p)$. This is impossible since, due to Theorem 4.11, $<^*_x$ is an order relation. So, $(\mathrm{rf}^i_h, p) \rightarrow_x (\mathrm{tf}^i_k, q)$ and then x satisfies Q.

Conversely, we suppose that x satisfies Q. Let us show that if $(P_h, p) <_x (P_k, q)$, then $(\mathrm{tf}^{h+1}_h, p) \Rightarrow_x (\mathrm{tf}^{k+1}_k, q)$.

$(P_h, p) <_x (P_k, q)$ implies $(\mathrm{tf}^i_h, p) \Rightarrow_x (\mathrm{rf}^i_k, q)$ (1) or $(\mathrm{rf}^i_h, p) \Rightarrow_x (\mathrm{tf}^i_k, q)$ (2), with $h = i-1$ and $k = i$ (3) or $h = i$ and $k = i-1$ (4).

In case (3), let us suppose that $(\mathrm{tf}^{i+1}_i, q) \rightarrow_x (\mathrm{tf}^i_{i-1}, p)$. So $(\mathrm{tf}^i_i, q) \rightarrow_x (\mathrm{tf}^i_{i-1}, p)$ and, therefore due to property Q, $(\mathrm{tf}^i_i, q) \rightarrow_x (\mathrm{rf}^i_i, q) \rightarrow_x (\mathrm{tf}^i_{i-1}, p)$.

This is incompatible with (1) and (2).

In case (4), let us suppose that $(\mathrm{tf}^i_{i-1}, q) \Rightarrow_x (\mathrm{tf}^{i+1}_i, p)$ (5). Then,

- if $(\mathrm{tf}^i_i, p) \rightarrow_x (\mathrm{tf}^i_{i-1}, q)$ then, due to property Q, we have $(\mathrm{tf}^i_i, p) \rightarrow_x (\mathrm{rf}^i_i, p) \rightarrow_x (\mathrm{tf}^i_{i-1}, q)$.

This is incompatible with (5).

- if $(\mathrm{tf}^i_{i-1}, q) \rightarrow_x (\mathrm{tf}^i_i, p)$, then, due to property Q, we have $(\mathrm{tf}^i_{i-1}, q) \rightarrow_x (\mathrm{rf}^i_{i-1}, q) \rightarrow_x (\mathrm{tf}^i_i, p)$.

This is incompatible with (1) and (2).

By extension, we have $(P_h, p) <^*_x (P_k, q)$ implies $(\mathrm{tf}^{h+1}_h, p) \Rightarrow_x (\mathrm{tf}^{k+1}_k, q)$.

Since $\Rightarrow_x$ is an order relation, so is $<^*_x$. Then, x is a correct computation.

References

[1] P.A. Bernstein and N. Goodman, Concurrency control in distributed data base systems. *Computing Surveys* **13** (2) (1981) 185-221.

[2] J. Büchi, On a decision method in restricted second-order arithmetic, in: *Internat. Congress Logic Method Phil. Sci.* (Standford Univ. Press, 1962).

[3] R. Cori and D. Perrin, Sur la reconnaissabilite dans les monoides partiellement commutatifs libres, Rapport de Recherche, Université de Bordeaux 1, 1984; *RAIRO Inform. Théorique*, to appear.

[4] E.W. Dijkstra, Hierarchical ordering of sequential process, *Acta Informatica* **1** (2) (1971) 115-138.

[5] K.P. Eswaran, J.N. Gray, R.A. Lorie and J.L. Traiger, The notions of consistency and predicate locks in data base systems, *Comm. ACM* **19** (11) (1976) 624-633.

[6] M.P. Flé and G. Roucairol, On serializability of iterated transactions, *ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, Ottawa, Canada (1982) 194-200.

[7] R.M. Karp and R.E. Miller, Parallel program schemata, *J. Comput. System Sci.* **3** (1969) 147-195.

[8] R.M. Keller, Parallel program schemata and maximal parallelism, *J. Assoc. Comput. Mach.* **20** (3) (1973) 514-537.

[9] C.H. Papadimitriou, P.A. Bernstein and J.B. Rothnie, Some computational problems related to data base concurrency control, *Proc. Conf. on Theoretical Computer Science*, Waterloo, Canada (1977) 272-282.

[10] G. Roucairol, Mots de synchronization, *RAIRO Informatique/Computer* **12** (4) (1978) 277-290.