Acta Informatica (Springer-Verlag) 32, pp.155-170, 1995

Efficient Algorithms for Checking the Atomicity of a Run of Read and Write Operations

Lefteris M. Kirousis[1,2,3] Andreas G. Veneris[4,5]

January 1994

Abstract

Let $X_1, \ldots, X_c$ be variables shared by a number of processors $P_1, \ldots, P_q$ which operate in a totally asynchronous and wait-free manner. An operation by a processor is either a write to one of the variables or a read of the values of *all* variables. Operations are *not* assumed to be instantaneous and may arbitrarily overlap in time. A succession of possibly overlapping operations $a_1, \ldots, a_n$ (i.e., a run) is said to be atomic, if these operations can be serialized in a way compatible with any existing precedences among them and so that any read operation returns for each variable the value of the most recent —with respect to the serialization— write operation on this variable. This paper examines the complexity of the combinatorial problem of testing a run for atomicity. First it is pointed out that when there is only one shared variable or when only one processor is allowed to write to each variable, known theorems lead to polynomial-time algorithms for checking the atomicity of a run (the variable of the time-complexity function is the number of operations in the run). It is then proved that checking atomicity has polynomial-time complexity in the general case of more than one variables and with all procesors allowed to read and write. For the proof, the atomicity problem is reduced to the problem of consecutive 1s in matrices. The reduction entails showing a combinatorial result which might be interesting on its own.

1 Introduction

A **shared array** is a data object comprising a number of variables $X_1, \ldots, X_c$ that are shared by a number of processors $P_1, \ldots P_q$. An operation by a processor is either a write to one of the variables of a value selected by the processor from a set of permissible values or a read of the values of *all* variables. Processors are assumed to operate totally asynchronously and in a wait-free fashion. Depending on the number of processors that are allowed to read or write

[1]Department of Computer Engineering and Informatics, University of Patras, Rion, 265 00 Patras, Greece (e-mail: kirousis@cti.gr).

[2]Computer Technology Institute, P.O. Box 1122, 261 10 Patras, Greece.

[3]The research of the first author was partially supported by the European Union ESPRIT Basic Research Projects ALCOM II (contract no. 7141) and Insight II (contract no. 6019).

[4]University of Illinois at Urbana-Champaign, Department of Computer Science, Room 2412 DCL, 1304 West Springfield Avenue, Urbana, Illinois 61801, USA (e-mail: veneris@uivlsi.csl.uiuc.edu).

[5]The research of the second author was carried out while he was a student at the University of Patras and also during subsequent visits of his to Patras.

and the number of variables, we get different types of shared arrays. (In the literature, shared arrays with a single variable are also known as **registers**, while shared arrays with more than one variables are also known as **composite registers**. In the latter case, the variables are also called **components** of the composite register.)

A succession of operations $a_1, \ldots, a_n$ executed by the processors is called a **run** or **history**.

We assume that there is a precedence relation among the operations of a run (denoted by '$\rightarrow$') which is a strict partial order. (A strict partial order is by definition a transitive and irreflexive relation. A relation R is irreflexive if for all a in its domain it is *not* true that aRa.) Two operations a and b are said to overlap if they are incomparable under the precedence relation '$\rightarrow$'. It is assumed that operations by the same processor are always comparable under '$\rightarrow$'.

We assume that for every read operation r and each variable X_i, $\; i = 1, \ldots, c$, there is a uniquely defined write operation on X_i that writes the value that r returns. Formally, we assume that there are reading functions $\pi_i, i = 1, \ldots c$ such that $\pi_i(r)$ writes the value that r returns from X_i. Notice that we do *not* assume that the functions π_i are "known" to the reader (such functions are defined also in [17, 19]).

It is sometimes convenient to think in terms of the universal time model. In this model, operations are assumed to have a duration in time represented by a closed interval (however, it is not assumed that processors can read the time). Moreover, for any two operations a and b, $a \rightarrow b$ iff the right end-point of the duration interval of a is strictly less than the left end-point of the duration interval of b (see [15, 16]). Notice that because the processors operate asynchronously and in a wait-free fashion, the duration intervals of operations by different processors may arbitrarily intersect and the length of the duration interval of an operation depends only on the speed of the executing processor (duration intervals of operations by the same processor do not intersect). In this paper however, neither in the statement of our results *nor* in their formal proofs do we make use of the universal time model.

A run of operations is called **atomic** if the partial order '$\rightarrow$' on the operations of the run can be extended to a strict *total* order '$\Rightarrow$' on them such that for each variable X_i, $i = 1, \ldots, c$ and for each read operation r of the run, $\pi_i(r) \Rightarrow r$ and, moreover, there is no write operation w on the variable X_i in the run so that $\pi_i(r) \Rightarrow w \Rightarrow r$. Informally, atomicity guarantees that the operations of a run can be serialized in a way compatible with any precedences among them and so that a read operation returns for each variable its most recent value (see [8, 16]).

The problem of designing shared arrays where all runs are guaranteed to be atomic (such arrays are called **atomic arrays**) is a much studied problem. In the literature, there exist many constructions for different types of atomic shared arrays (single- or multi-variable, single- or multi-reader, single- or multi-writer *etc*) using as building blocks elementary variables that satisfy conditions much weaker than atomicity (among others see, e.g., [1, 2, 6, 7, 9, 10, 11, 13, 17, 19, 20, 21, 22]).

In this paper, we examine the complexity of the combinatorial problem of checking the atomicity of a run on a shared array not known to be atomic. We call this problem the 'atomicity problem'. Formally, an instance of the problem consists of:

- A class $\{a_1, \ldots, a_n\}$ of entities, called "operations", partitioned into $c+1$ subclasses ($1 \leq c < n$; the integer c is called the "number of shared variables"). The subclasses are: (i) a subclass of operations called "read operations" and (ii) c subclasses of operations called "write operations on the variable i", $i = 1, \ldots, c$. Each of the above mutually disjoint $c+1$ subclasses is assumed to contain at least one operation.

- A strict partial order '$\to$' on the set of operations.

- For each $i = 1, \dots c$, a function π_i from the subclass of read operations to the subclass of write operations on the variable i.

- A class $\{P_1, \dots, P_q\}$ of entities, called "processors" ($1 \leq q \leq n$). Each operation a_i is assigned to a unique processor, called the processor that "executes a_i". Operations executed by the same processor are assumed to be always comparable under '$\to$'. Any processor may read and/or write any variable.

The **atomicity problem** is the question of whether there exists a strict total order $\Rightarrow$ on the operations $a_1, \dots, a_n$ such that:

- (i) for any two operations a_i and a_j if $a_i \to a_j$ then $a_i \Rightarrow a_j$ and

- (ii) for any read operation r and for any $i = 1, \dots, c$, $\pi_i(r) \Rightarrow r$ and, moreover, there is no write operation w on the variable i such that $\pi_i(r) \Rightarrow w \Rightarrow r$.

An instance of the atomicity problem is called a **run** or **history**. Of course, terms such as "operation", "processor" *etc* have no other semantics in the context of the combinatorial problem other than what is described in the above definition of a run.

The size of the problem is assumed to be the number of operations n. We asume that to check whether an operation is a read operation or a write operation on some i, to compute the image of a read operation under any of the π_is and to compare two operations with respect to $\to$ all have constant cost.

The atomicity problem is sometimes solved by supplying necessary and sufficient conditions (**criteria**) for a run to be atomic. The time complexity of such criteria is the time necessary to check whether they hold. Notice that to find necessary and sufficient conditions for a program so that its output is always an atomic run is a totally different problem [3]. The atomicity problem in the sense considered in this paper refers only to the run and not to the program that outputs it.

If $c = 1$, the run is called a run on a single variable. If $c \geq 1$, the run is called a run on a multi-variable array. If all read operations of a run are executed by the same processor (in which case, by assumption, they are linearly ordered by '$\to$'), the run is called a single-reader run. In the absence of such a restriction, it is called a multi-reader run. If for each $i = 1, \dots, c$ the write operations on the variable i are executed by the same processor (in which case, by assumption, the write operations on each i are linearly ordered by '$\to$'), the run is called a single-writer-per-variable run. In the absence of such a restriction, it is called a multi-writer-per-variable run.

Obviously, the above combinatorial problem is not explicitly related to the classical problem of constructing atomic shared arrays from elementary variables. However, because of the fact that a major difficulty with the extant constructions of atomic registers is proving their correctness, we believe that efficient algorithms for testing a run for atomicity can lead to correctness-proof strategies. This essentially is the case with the results in [4], where the correctness of the now classical Vitányi-Awerbuch matrix protocol [22] is proved by first giving a simple atomicity criterion. A similar approach is also used in [19]. Those papers, however, do not consider the problem of checking atomicity from a complexity-theory point of view. Furthermore, it is plausible that simple atomicity criteria of combinatorial nature may lead to new constructions for atomic registers or at least make more understandable the existing ones. Indeed, in the search of novel (presumably simpler than the extant ones) atomic register constructions, it suffices to aim only at ensuring that the corresponding combinatorial atomicity criterion is satisfied (this was the case with the construction in [12]).

For runs on a single variable, existing theorems lead easily to polynomial-time atomicity criteria. For multi-variable runs which have a single writer per variable, we can again easily derive a polynomial-time criterion for atomicity. For the case of a multi-variable, multi-writer-per-variable, but single-reader run, an atomicity criterion is given in [14]. These results are outlined in Section 2.

In the general case, however, of a multi-variable, multi-reader, multi-writer-per-variable run none of the methods used in the results above seem to lead to an efficient criterion (in [2], a necessary and sufficient condition for atomicity of a run is given, however that condition entails showing the existence of total orders on subclasses of operations, so combinatorially, it leads to exponential-time algorithms). In Section 3, we give a polynomial-time criterion for atomicity for the general case by reducing the atomicity problem to the problem of consecutive 1s in matrices. For the reduction, we prove a combinatorial result which might be interesting on its own.

2 Special Cases

In this section, we explain how known theorems lead to efficient algorithms for checking the atomicity of a run in certain special cases. To our knowledge, the atomicity problem has not been studied before from a complexity-theory point of view (in contrast to the serializability problem for transactions on a database, see e.g. [18]). Detailed proofs are not given, since what is proved in the next section, which is the main contribution of this paper, covers as special cases all the results of this section.

2.1 Run on a Single Variable

In this subsection we assume that $c = 1$, i.e., we assume that the run is on a single variable. The reading function for this variable is denoted by π.

If, in addition, we assume that we have a single-writer run, it is essentially proved in [16] that the run is atomic if and only if the following conditions hold:

- For any read operation r, not$(r \rightarrow \pi(r))$.
- For any read operation r, it is not the case that there exists a write operation w such that $\pi(r) \rightarrow w \rightarrow r$.
- For any two read operations r_1 and r_2, if $r_1 \rightarrow r_2$ then it is not the case that $\pi(r_2) \rightarrow \pi(r_1)$.

Obviously, to check the truth of the above conditions requires polynomial (in the number of operations) time.

For the more general case of a multi-reader, multi-writer run (on a single variable), an efficient atomicity criterion is given in [4] (related results are also given in [19]). Awerbuch *et al.* define for each write operation w its **clan** (denoted by $[w]$) to be the class $\{w\} \cup \{r : r \text{ is a read operation and } \pi(r) = w\}$. In other words, a clan is a write operation together with all read operations (if any) that read it.

They then define a relation $\rightarrow^{\pi}$ among clans as follows:

$$[w]\rightarrow^{\pi}[w'] \text{ iff } \exists \text{ operation } a \in [w] \text{ and } \exists \text{ operation } a' \in [w'] \text{ such that } a \rightarrow a'.$$

They prove in the sequel that the run is atomic if and only if the following two conditions hold:

- For any read operation r, not$(r \rightarrow \pi(r))$.
- The relation $\rightarrow^{\pi}$ is acyclic.

Since looking for cycles in a relation is a well-known problem that can be efficiently solved (e.g. by reducing it to the matrix-multiplication problem), it is not hard to see that the criterion of Awerbuch *et al.* gives an efficient algorithm for testing a run for atomicity.

2.2 Run on a Multi-Variable Shared Array

In this subsection we allow c to take any integer value ≥ 1.

In [2] , a condition for the atomicity of a run is given. It is proved that a run is atomic if for each $i = 1, \ldots, c$, there exists a function ϕ_i mapping every read operation and every write operation on the variable i to some natural number so that the following conditions are satisfied:

- For each pair of distinct write operations v and w on a variable i, $\phi_i(v) \neq \phi_i(w)$. Furthermore, if $v \rightarrow w$, then $\phi_i(v) < \phi_i(w)$.
- For each read operation r and for each $i = 1, \ldots, c$, there exists a write operation w on the variable i such that $\phi_i(r) = \phi_i(w)$. Furthermore, the value that r reads from the variable i is equal to the value that w writes.
- For each read operation r and each write operation w on a variable i, if $r \rightarrow w$ then $\phi_i(r) < \phi_i(w)$ and if $w \rightarrow r$ then $\phi_i(w) \leq \phi_i(r)$.
- For each pair of read operations r and s if $(\exists i)(\phi_i(r) < \phi_i(s))$ or if $r \rightarrow s$, then $(\forall i)(\phi_i(r) \leq \phi_i(s))$.
- For each read operation r, each write operation v on a variable i and each write operation w on a variable j $(i, j = 1, \ldots, c)$, if $v \rightarrow w$ and $\phi_j(w) \leq \phi_j(r)$, then $\phi_i(v) \leq \phi_i(r)$.

Although, only the sufficiency of the above conditions for showing atomicity is proved in [2], it is not difficult to see that they are necessary as well. Notice that the existence of the functions ϕ_i amounts to assuming that write operations on each variable can be linearly ordered in a way compatible with the semantics of shared arrays. Therefore, in the case of multi-writer-per-variable runs, this criterion, from a combinatorial complexity point of view, cannot be more useful than the initial definition of atomicity.

However, for the case of a single-writer-per-variable run we have:

Theorem 1 *A multi-variable, multi-reader but single-writer-per-variable run is atomic iff the following conditions hold:*

- *For any read operation r and for any $i = 1, \ldots, c$, it is not the case that $r \rightarrow \pi_i(r)$.*
- *For any read operation r, for any $i = 1, \ldots, c$ and for any write operation w on the variable i it is not the case that $\pi_i(r) \rightarrow w \rightarrow r$.*

- *For any read operations r_1, r_2, and for any $i, j = 1, \ldots, c$, not both $\pi_i(r_1) \to \pi_i(r_2)$ and $\pi_j(r_2) \to \pi_j(r_1)$ can hold. Also, not both $r_1 \to r_2$ and $\pi_i(r_2) \to \pi_i(r_1)$ can hold.*
- *For any read operation r, for any $i, j = 1, \ldots, c$ and for any write operation w on the variable i, it is not the case that $\pi_i(r) \to w \to \pi_j(r)$.*

Sketch of Proof The necessity of the conditions is easy to prove. For the sufficiency, we make use of Anderson's result stated above. Define for a write operation w on a variable i the value $\phi_i(w)$ to be the number of write operations on the variable i that precede w with respect to the relation '$\to$' (notice that, since by assumption we have a single-writer-per-variable run, the write operations on each variable are linerly ordered by '$\to$'). Moreover, for a read operation r, define $\phi_i(r)$ to be $\phi_i(\pi_i(r))$. It is easy to check that the conditions of Anderson's criterion are satisfied. $\square$

It is obvious that to check the conditions of the above theorem requires polynomial time.

A restricted version of the above theorem, for the case of single-reader runs, appears in [12]. The above more general version of it probably has been known, although, to our knowledge, has not appeared anywhere.

Finally, for the case of multi-writer-per-variable but single-reader runs, a necessary and sufficient condition for atomicity is given in [14]. It is proved in that paper that a multi-writer-per-variable but single-reader run is atomic if and only if it is *not* possible to have two sequences $r_1, \ldots, r_m$ and $w_0, \ldots w_m$ of m read and $m+1$ write operations (m is any integer ≥ 1), such that if i_j denotes the variable of the operation w_j ($j = 0, \ldots, m$) then all the following hold:

- $w_0 \stackrel{=}{\to} w_1 \stackrel{=}{\to} \cdots \stackrel{=}{\to} w_{m-1} \stackrel{=}{\to} w_m$.
- $\pi_{i_1}(r_1) = \pi_{i_1}(r_2)$ and $\pi_{i_2}(r_2) = \pi_{i_2}(r_3)$ and $\cdots$ and $\pi_{i_{m-1}}(r_{m-1}) = \pi_{i_{m-1}}(r_m)$.
- Either $r_1 \to w_0$ or ($\exists$ read operation r)($r_1 \stackrel{=}{\to} r$ and $\pi_{i_0}(r) \stackrel{=}{\to} w_0 \neq \pi_{i_0}(r_1)$).
- Either $w_m \to r_m$ or ($\exists$ read operation r')($r' \stackrel{=}{\to} r_m$ and $w_m \stackrel{=}{\to} \pi_{i_m}(r')$).

Above, '$a \stackrel{=}{\to} b$' stands for '$a \to b$ or $a = b$'.

Observe that the above condition does not introduce as a requirement the existence of any linear orders. Essentially for this reason, although it does not immediately lead to an efficient algorithm, it can be modified to do so. We avoid giving the details, since in the next section we describe an efficient algorithm for the most general case.

3 The General Case

This section contains the main technical contribution of the paper. We give a characterization of atomicity of a multi-reader, multi-writer-per-variable run on a multi-variable shared array. The characterization yields a polynomial-time algorithm for checking atomicity. In the sequel, assume that ρ denotes an instance of the combinatorial atomicity problem with precedence relation $\to$ and reading functions $\pi_i, i = 1, \ldots, c$. As a first approach towards obtaining an efficient criterion for the atomicity of ρ, we find certain *only necessary* conditions that '$\to$' must satisfy in order to be extendible to a total order $\Rightarrow$ with the properties required in the definition of atomicity. For that, we define a relation that by definition will *extend* the relation '$\to$' (we denote this extension by '$\stackrel{\text{xt}}{\to}$'), but will *not* in general be acyclic. Now, if ρ is atomic, it turns out that '$\stackrel{\text{xt}}{\rightarrow}$' must be a subset of '$\Rightarrow$'. Therefore, in case ρ is atomic, '$\stackrel{\text{xt}}{\rightarrow}$' must be acyclic (or equivalently, since by definition will be transitive, it must be irreflexive).

Formally, we first define:

Definition 1 *Given a write operation w on a variable i, the* **clan** *of w (denoted by C_w) is the set:*

$$C_w = \{w\} \cup \{r : \pi_i(r) = w\},$$

in other words the clan of a write operation w consists of w and all read operations (if any) that read w.

The notion of a clan was introduced in [4]. Notice that for each variable $i = 1, \ldots, c$ separately, the clans of the write operations on the variable i partition the read operations of the run into disjoint sets.

We now define the relation '$\stackrel{\text{xt}}{\rightarrow}$':

Definition 2 *If a and b denote arbitrary operations, define* recursively $a \stackrel{\text{xt}}{\rightarrow} b$ *iff at least one of the following conditions hold:*

- *1. $a \rightarrow b$.*
- *2. For some read operation r and for some variable i, $a = \pi_i(r)$ and $b = r$.*
- *3.* $\exists$ operation $c : a \stackrel{\text{xt}}{\rightarrow} c$ and $c \stackrel{\text{xt}}{\rightarrow} b$.
- *4. There exist two* different *write operations v and w on the same variable i, such that $a \in C_v$ and $b \in C_w$ and for some $a' \in C_v$ and $b' \in C_w$ we have that $a' \stackrel{\text{xt}}{\rightarrow} b'$.*

Observation *It is trivial to see that because of condition 3 in the above definition, the relation '$\stackrel{\text{xt}}{\rightarrow}$' is transitive.*

Informal Remark 1 The relation '$\stackrel{\text{xt}}{\rightarrow}$' is defined with the purpose to contain *exactly all* pairs (a, b) of operations such that in the final serialization, b must necessarily be placed after a for reasons dictated by the precedences in '$\rightarrow$'. To explain why we had to give a recursive definition, consider the following example: Let r_1, r_2 and and r_3 be three read operations and w_1 and w_2 be two write operations on variables i and j, respectively, such that the *only* precedences among these five operations are given by $r_1 \rightarrow w_1 \rightarrow w_2$. Assume, moreover, that $\pi_i(r_1) = \pi_i(r_2)$ and $\pi_j(r_2) = \pi_j(r_3)$. Now, because $r_1 \rightarrow w_1$ and $\pi_i(r_1) = \pi_i(r_2)$ and since w_1 writes to the variable i, we can easily conclude by case analysis that in the final atomic serialization of the operations, w_1 must be placed after r_2. Now given this fact and because $\pi_j(r_2) = \pi_j(r_3)$ and also because w_2 writes to the variable j, we conclude that in the final serialization w_2 must be placed after r_3. However, because originally the only precedences among the five operations were that $r_1 \rightarrow w_1 \rightarrow w_2$, the conclusion that w_2 must be placed after r_3 could not have been reached without first showing that w_1 must be placed after r_2. Examples of the same nature can be given with arbitrarily long sequences of reads and writes. So if the relation '$\stackrel{\text{xt}}{\rightarrow}$' is to contain as many pairs as possible, it must be defined recursively (or, alternatively, be defined as a fixed point of some iteration).

Informal Remark 2 If we assume the truth of our informal claim that '$\stackrel{\text{xt}}{\rightarrow}$' does indeed contain *exactly all* pairs that it should contain because of the precedences in '$\rightarrow$', one might tempted to argue that if '$\xrightarrow{\text{xt}}$' is irreflexive, i.e., if it is extendible to a strict total order (notice that it is transitive by definition), then an atomic serialization exists. Unfortunately, this is not true as the following example shows: Let r_1, r_2 and r_3 be three read operations and let i,j and k be three distinct variables such that: $\pi_i(r_1) = \pi_i(r_2) \neq \pi_i(r_3)$ and $\pi_j(r_2) = \pi_j(r_3) \neq \pi_j(r_1)$ and $\pi_k(r_3) = \pi_k(r_1) \neq \pi_k(r_2)$. Also assume that originally there are *no* precedences at all among the above nine read and write operations (the number of these operations is nine and not twelve, because of the three pairs of equalities among them). Now it can be easily seen that the relation '$\xrightarrow{\text{xt}}$' will only contain the nine pairs $\pi_i(r_l) \xrightarrow{\text{xt}} r_l, \pi_j(r_l) \xrightarrow{\text{xt}} r_l, \pi_k(r_l) \xrightarrow{\text{xt}} r_l, l = 1, 2, 3$ and nothing else. Therefore, '$\xrightarrow{\text{xt}}$' is irreflexive. However, an easy case analysis shows that these nine operations cannot be atomically serialized. (Notice that in the universal time model, if we further demand that '$\xrightarrow{\text{xt}}$' should be an interval order whose intervals are obtained by shrinking the duration intervals of the nine given operations, then it cannot be irreflexive anymore. From a complexity point of view though, this fact is not of much help, since we would have to examine all possible shrinkings.) The impossibility of the serialization is due to the fact that none of the three read operations r_1, r_2, r_3 can be placed between the other two. This example indicates that to get a set of conditions sufficient for atomicity we must include not only the irreflexivity of '$\xrightarrow{\text{xt}}$', but also that read operations which read the same write operation should be consecutive in their final serialization (not counting write operations on other variables that may interfere). Fortunately, it turns out that these two constraints, namely the irreflexivity of '$\xrightarrow{\text{xt}}$' and the consecutive reads property are indeed sufficient for atomicity and can be efficiently checked. It is interesting to note that it is only for multi-reader, multi-writer-per-variable runs on multi-variable shared arrays that the consecutive reads condition is needed. If one (or more) of these three "multi-" prefixes is substituted by the prefix "single-" then, by the results in the previous section, the consecutive reads constraint is a consequence of the irreflexivity of '$\xrightarrow{\text{xt}}$'.

We now return to our formal results.

Theorem 2 *Any strict linear order that satisfies the atomicity requirements for the run ρ must necessarily contain as subset the relation '$\xrightarrow{\text{xt}}$'.*

Proof Let $\Rightarrow$ be a strict total order satisfying the atomicity requirements for ρ. In other words, we assume that

- (i) '$\Rightarrow$' extends '$\rightarrow$',
- (ii) for any read operation r and any $i = 1, \ldots, c$, $\pi_i(r) \Rightarrow r$ and
- (iii) if r is a read operation and $i = 1, \ldots, c$, then there is no write operation on the variable i such that $\pi_i(r) \Rightarrow w \Rightarrow r$.

The proof that $\xrightarrow{\text{xt}} \subseteq \Rightarrow$ is by induction on the recursive definition of '$\xrightarrow{\text{xt}}$'. We distinguish the following four cases:

- 1. Suppose that $a \xrightarrow{\text{xt}} b$ because $a \rightarrow b$ (i.e., because of condition 1 of Definition 2). Then, since '$\Rightarrow$' extends '$\rightarrow$', we have that $a \Rightarrow b$.
- 2. Suppose that $a \xrightarrow{\text{xt}} b$ because there is a read operation r and a variable i such that $a = \pi_i(r)$ and $b = r$ (i.e., because of condition 2 of Definition 2). Then from assumption (ii) above we have that $a \Rightarrow b$.

- 3. Suppose that $a \stackrel{\text{xt}}{\rightarrow} b$ because $\exists$ operation c such that $a \stackrel{\text{xt}}{\rightarrow} c$ and $c \stackrel{\text{xt}}{\rightarrow} b$ (i.e., because of condition 3 of Definition 2). Then by the inductive hypothesis $a \Rightarrow c$ and $c \Rightarrow b$. Therefore from the transitivity of '$\Rightarrow$' it follows that $a \Rightarrow b$.

- 4. Suppose that $a \stackrel{\text{xt}}{\rightarrow} b$ because there exist two *different* write operations v and w on the same variable i, such that $a \in C_v$ and $b \in C_w$ and for some $a' \in C_v$ and $b' \in C_w$ $a' \stackrel{\text{xt}}{\rightarrow} b'$ (i.e., because of condition 4 of Definition 2). To prove that $a \Rightarrow b$, we distinguish several cases with respect to the type of operations a, a', b and b'. Suppose that a, a' and b' are read operations, while $b = \pi_i(b')$. By the induction hypothesis $a' \Rightarrow b'$. Therefore, by assumption (ii) above, we have that $\pi_i(a') \Rightarrow b'$. Also, since $a, a' \in C_v$ and $b, b' \in C_w$, we have that $\pi_i(a) = \pi_i(a') \neq \pi_i(b') = b$. Therefore, by assumption (iii) above, $\pi_i(a) \Rightarrow b$ (indeed, otherwise we would have $\pi_i(b') = b \Rightarrow \pi_i(a) = \pi_i(a') \Rightarrow b'$). Now, again by assumption (iii), we conclude that $a \Rightarrow b$ (indeed, otherwise we would have $\pi_i(a) \Rightarrow b \Rightarrow a$). All other cases concerning the type of the operations a, a', b and b' are handled similarly. $\square$

As we mentioned in Informal Remark 1, the acyclicity of '$\stackrel{\text{xt}}{\rightarrow}$' is not a sufficient condition for the atomicity of ρ. We now introduce another necessary condition that guarantees that in the final serialization, read operations of the same clan will be consecutive (not counting write operations on other variables that may interfere).

First a definition:

Definition 3 *Given a run ρ, the writes*-vs-*reads matrix of ρ is the matrix whose rows represent the write operations of ρ (write operations on all variables are considered), its columns represent the read operations of ρ and the value of an entry at row w and column r is equal to one if $\pi_i(r) = w$ (assuming that w is a write operation on the variable i) and is equal to zero, otherwise.*

The consecutive 1s problem for matrices is an old combinatorial problem whose input instances are matrices with entries 0 or 1 and whose output is a rearrangement of the columns of the matrix so that in each row of the rearranged matrix there is no 0 between two 1s. If the columns of a matrix can thus be rearranged, we say that it has the consecutive 1s property. There are linear time algorithms for checking the consecutive 1s property ([5]).

We now state and prove the following theorem:

Theorem 3 *The writes*-vs-*reads matrix of an atomic run has the consecutive 1s property.*

Proof Let $\Rightarrow$ be an atomic linear order on the set of operations of ρ. Rearrange the columns of the writes-*vs*-reads matrix so that a column corresponding to a read r_1 (column r_1, for short) is to the left of another column r_2 iff $r_1 \Rightarrow r_2$. We claim that thus in every row, the 1s are consecutive. Indeed, if this is not the case for a row corresponding to a write operation w on, say, a variable i, then there must exist three consecutive columns, say r_1 r_2 and r_3, with entries on the row w: 1, 0 and 1, respectively. But then we would have that $\pi_i(r_1) = \pi_i(r_3) = w \neq \pi_i(r_2)$, while $\pi_i(r_1) \Rightarrow \pi_i(r_2) \Rightarrow \pi_i(r_3)$. It can be immediately verified, by case analysis, that this cannot be so, since '$\Rightarrow$' is assumed to satisfy the definition of atomicity. $\square$

We now state our most important result:

Theorem 4 *A run ρ is atomic if and only if the relation '$\stackrel{\text{xt}}{\rightarrow}$' is irreflexive and the writes*-vs-*reads matrix of the run has the consecutive 1s property.*

The necessity of the conditions of Theorem 4 follow easily from Theorems 2 and 3. The converse is an immediate corollary of the two lemmas that follow. The first lemma pertains to the properties of atomicity, while the second is essentially a result of combinatorics. Before the Lemmas, we give a definition.

Definition 4 *A write operation w on a variable i is called* **visible** *if there is a read operation r such that $\pi_i(r) = w$. Otherwise, the write operation is called* **invisible**.

We now state and prove the Lemmas.

Lemma 1 *If the relation '$\xrightarrow{\text{xt}}$' of a run ρ is irreflexive and if the columns of the writes*-vs-*reads matrix of ρ can be rearranged in such a way as (i) the 1s in all rows are consecutive and (ii) for any two read operations r and r' for which $r \xrightarrow{\text{xt}} r'$ the column corresponding to r is to the left of the column corresponding to r', then ρ is atomic.*

Proof of Lemma 1 Let $r_1, \ldots r_k$ be the enumeration from left to right of the columns of the writes-*vs*-reads matrix in a rearrangement that satisfies the conditions of Lemma 1. Let S_0 denote the sequence $r_1, \ldots, r_k$. We are going to extend the sequence S_0 to a sequence of all operations of the run so that the linear order in which the operations appear in the extended sequence satisfies the atomicity requirements. So, let $w_1, \ldots, w_l$ be the write operations of the run (taken in any order). Let a be the leftmost element of the sequence S_0 for which $w_1 \xrightarrow{\text{xt}} a$. Extend S_0 to a sequence S_1 that contains w_1 by placing w_1 immediately to the left of a (if no such a exists then place w_1 to the right end of the sequence S_0). Repeat the same procedure for w_2 and S_1 to obtain a sequence S_2, and so on recursively, until we get a sequence S_l containing all operations of the run. We now make and prove three claims:

- *Claim 1* For any S_j and any operations a and b in S_j, if $a \xrightarrow{\text{xt}} b$, then in S_j, a lies to the left of b, i.e. all S_js are compatible with '$\xrightarrow{\text{xt}}$'.

Proof of Claim 1 From the hypotheses of Lemma 1, Claim 1 holds for S_0. Suppose, for an inductive proof, that Claim 1 is true for $S_1, \ldots, S_{j-1}$ ($j \geq 1$) and suppose, towards a contradiction, that it is not true for S_j. Then there must be an element b in S_j lying to the right of w_j such that $b \xrightarrow{\text{xt}} w_j$. Also, if w_j^+ is the element of S_j immediately to the right of w_j, then by the construction of S_j, $w_j \xrightarrow{\text{xt}} w_j^+$ and, moreover, w_j^+ lies to the left of (or coincides with) b. But then, by the transitivity of '$\xrightarrow{\text{xt}}$', we have that $b \xrightarrow{\text{xt}} w_j^+$. On the other hand, both b and w_j^+ are elements of S_{j-1}, which, by the induction hypothesis, is compatible with '$\xrightarrow{\text{xt}}$'. Therefore, we have a contradiction and so Claim 1 is proved (if $b = w_j^+$, then the contradiction is due to the irreflexivity of '$\xrightarrow{\text{xt}}$').

- *Claim 2* Let w be a write operation in some S_j. Given that in S_j there exist read operations lying to the *right* of w, let r_w denote that one among them that is closest to w. Then: (i) $w \xrightarrow{\text{xt}} r_w$ and (ii) if w is visible and if i denotes the variable that w writes to, then $\pi_i(r_w) = w$.

Proof of Claim 2 Claim 2 is vacuously true for S_0. Suppose, for an inductive proof, that it is also true for $S_1, \ldots, S_{j-1}$ ($j \geq 1$). Because of the induction hypothesis, we only have to prove Claim 2 for the element $w_j \in S_j$. Let w_j^+ be the element of S_j immediately to the right of w_j and let r_{w_j}

be the read operation that among all read operations lying to the right of w_j is closest to w_j (if the operation w_j^+ or the operation r_{w_j} cannot be defined, then there is nothing to prove). Then by the construction of S_j, $w_j \stackrel{\mathrm{xt}}{\rightarrow} w_j^+$. If w_j^+ is a read operation, then $w_j^+ = r_{w_j}$ and so the first part of Claim 2 is proved. If, on the other hand, w_j^+ is a write operation, then by the induction hypothesis, $w_j^+ \stackrel{\mathrm{xt}}{\rightarrow} r_{w_j}$ and so by the transitivity of '$\stackrel{\mathrm{xt}}{\rightarrow}$', the first part of Claim 2 is proved again. To show the second part, let i_j denote the variable of the write operation w_j, let w'_j denote $\pi_{i_j}(r_{w_j})$ and assume, towards a contradiction, that w_j is visible and that $w'_j \neq w_j$. Let s be any read operation such that $\pi_{i_j}(s) = w_j$. It follows from Claim 1 and from the definition of '$\stackrel{\mathrm{xt}}{\rightarrow}$' that in S_j, s lies to the right of w_j. If $C_{w'_j}$ and C_{w_j} denote the clans of w'_j and w_j, respectively, then by the already proved first part of Claim 2, $C_{w_j} \ni w_j \stackrel{\mathrm{xt}}{\rightarrow} r_{w_j} \in C_{w'_j}$. Therefore, from the definition of '$\stackrel{\mathrm{xt}}{\rightarrow}$', we get that $C_{w_j} \ni s \stackrel{\mathrm{xt}}{\rightarrow} r_{w_j} \in C_{w'_j}$. ¿From this and Claim 1, it follows that, in S_j, r_{w_j} lies further than s to the right of w_j. This is a contradiction because of the way that r_{w_j} was defined. So, Claim 2 is proved.

- *Claim 3* For any S_j, any read operation r and any $i = 1, \ldots, c$ if both r and $\pi_i(r)$ are elements of S_j, then, in S_j, there is no write operation w' on the variable i which lies between $\pi_i(r)$ and r. (Observe that, in S_j, $\pi_i(r)$ lies to the left of r, because $\pi_i(r) \stackrel{\mathrm{xt}}{\rightarrow} r$ and also because S_j is, by Claim 1, compatible with '$\stackrel{\mathrm{xt}}{\rightarrow}$').

Proof of Claim 3 Assume that Claim 3 is not true and let r be a read operation and w and w' be different write operations both on the variable i so that (i) w, w' and r are all in S_j, (ii) $\pi_i(r) = w$ and (iii) in S_j, w lies to the left of w', which, in turn, lies to the left of r. Let also r_w (respectively, $r_{w'}$) be the read operation that among read operations is closest from the right to w (respectively, w'). Then in S_j —and therefore in S_0 as well—, r_w lies to the left of (or coincides with) $r_{w'}$ which, in turn, lies to the left of (or coincides with) r. Since $\pi_i(r) = w$, w is a visible write operation, so by Claim 2, $\pi_i(r_w) = w$. Therefore, by (ii) above, we get that $\pi_i(r_w) = \pi_i(r)$. Now, notice that from the hypotheses of Lemma 1 and the definition of S_0 it follows that if the columns of the writes-*vs*-reads matrix of the run are arranged in the order that they appear in S_0, then the matrix has the consecutive 1s property. Therefore, since, in S_0, $r_{w'}$ lies between r_w and r, we get that $\pi_i(r_{w'}) = \pi_i(r_w) = \pi_i(r) = w$. Therefore $r_{w'}$ belongs to the clan C_w. Also, from Claim 2 we have that $w' \stackrel{\mathrm{xt}}{\rightarrow} r_{w'}$. Now, since $w \neq w'$ and $C_{w'} \ni w' \stackrel{\mathrm{xt}}{\rightarrow} r_{w'} \in C_w$, we get from the definition of '$\stackrel{\mathrm{xt}}{\rightarrow}$' that $C_{w'} \ni w' \stackrel{\mathrm{xt}}{\rightarrow} w \in C_w$. However, this last relation contradicts Claim 1, because we have assumed that, in S_j, w lies to the left of w'. This concludes the proof of Claim 3.

Lemma 1 now is an immediate corollary of Claim 1 and Claim 3 (it suffices to serialize all operations in the order they appear in the last sequence S_l). □

Lemma 2 *If for a run ρ the relation '$\stackrel{\mathrm{xt}}{\rightarrow}$' is irreflexive and if the writes-*vs*-reads matrix of ρ has the consecutive 1s property, then there is a rearrangement of the columns of the writes-*vs*-reads matrix so that for this rearrangement as well, the 1s in all rows are consecutive and, moreover, for any two read operations r and r' for which $r \stackrel{\mathrm{xt}}{\rightarrow} r'$ the column corresponding to r is to the left of the column corresponding to r'.*

The proof of Lemma 2 entails showing a nontrivial Combinatorial Lemma that is stated and proved below (the proof of Lemma 2 follows the proof of the Combinatorial Lemma). Informally, this Combinatorial Lemma gives a sufficient (and obviously necessary as well) condition under which we can rearrange the elements of a totally ordered set P so that two things are accomplished: (i) the new total order obtained from the rearrangement extends a given partial order on P, which, in general, is unrelated to the initial total order of P and (ii) any given intervals of P remain intervals under the new total order. (Recall that if $\mathcal{P} = (P, <_{\rm tt})$ is a totally ordered set, then a subset I of P is called an **interval** with respect to $<_{\rm tt}$ (notationally, $<_{\rm tt}$-interval) if $(\forall p, q \in I)(\forall r \in P)(\text{if} \;\; p <_{\rm tt} r <_{\rm tt} q \;\; \text{then} \;\; r \in I)$.) To our knowledge, this result has not been proved before. We believe that it is interesting on its own both for what it states and its proof.

Combinatorial Lemma *Let $\mathcal{P} = (P, <^0_{\rm tt})$ be a finite* totally *ordered set ($<^0_{\rm tt}$ is a* strict *total order); let $I_j, j = 1, \dots l$ be a family of intervals with respect to $<^0_{\rm tt}$ and let $<_{\rm pr}$ be a strict* partial *order on the set P ($<_{\rm pr}$ is not in general a subset of $<^0_{\rm tt}$). Suppose that for any I_j and any $p \in P$ that does* not *belong to I_j the following two conditions hold:*

- *(i) if $(\exists q \in I_j)(q <_{\rm pr} p)$ then $(\forall q \in I_j)(q <_{\rm pr} p)$ and*
- *(ii) if $(\exists q \in I_j)(p <_{\rm pr} q)$ then $(\forall q \in I_j)(p <_{\rm pr} q)$.*

Then there exists a strict total order $<^1_{\rm tt}$ on P which extends $<_{\rm pr}$ and with respect to which all the I_js are still intervals.

Proof of the Combinatorial Lemma We first give some definitions. A subfamily of a given family of intervals (with respect to a total order) is called a **clique** if the elements of the subfamily have a nonempty intersection (this intersection is called the intersection of the clique). Notice that the requirement that the intervals of a clique have nonempty intersection is equivalent to the requirement that they pairwise have nonempty intersection. A clique which is not a proper subfamily of any other clique of the given family is called a **maximal clique**.

We will prove the Combinatorial Lemma by induction on the number of elements in P. If P has only one element, then it is trivially true. Assume that it is true for all sets with cardinality $\leq n$ and let P be a set with cardinality $n+1$. First delete from the class of intervals $I_j, j = 1, \dots, l$ the ones that have cardinality < 2 (observe that a total order '$<^1_{\rm tt}$' that extends '$<_{\rm pr}$' and satisfies the conclusions of the Combinatorial Lemma for this reduced class of intervals also satisfies the same conclusions for the initial class because singletons are intervals with respect to any order). For notational convenience, we denote this reduced class by $I_j, j = 1, \dots, l$ as well. We distinguish two cases:

Case I Assume that the family of $<^0_{\rm tt}$-intervals $I_j, j = 1, \dots, l$ has a maximal clique C with two or more elements in its intersection $\cap C$. Let p be an arbitrary element in $\cap C$. Consider the set $P^- = P - \{p\}$, the restrictions of the orders '$<^0_{\rm tt}$' and '$<_{\rm pr}$' to P^-, and the intersections $I_j \cap P^-, j = 1, \dots, l$. Once the hypotheses of the Combinatorial Lemma are true for a totally ordered set $\mathcal{P} = (P, <^0_{\rm tt})$, a partial order $<_{\rm pr}$ on P and a family of $I_j, j = 1, \dots, l$ of subsets of P, they obviously remain true for any suborder of $\mathcal{P}$ together with the corresponding restriction of the partial order $<_{\rm pr}$ and the corresponding restrictions of the sets $I_j, j = 1, \dots, l$. Therefore, we can legitimately apply the induction hypothesis to obtain a total order on P^- satisfying the conclusions of the Combinatorial Lemma for P^- and its corresponding orders and intervals. Let $p_1, \dots, p_n$ be the elements of P^- placed from left to right in a small-first fashion with respect to this total order on P^-, which was obtained by applying the induction hypothesis. It easily follows that the set $\cap C - \{p\}$, which is $\neq \emptyset$, is an interval with respect to this inductively defined order. Let $p_r, \dots, p_s$ $(1 \leq r \leq s \leq n)$ be the elements of $\cap C - \{p\}$. Let p_i be the leftmost element in the sequence $p_r, \ldots, p_s$ for which $p <_{\text{pr}} p_i$. Extend the sequence $p_1, \ldots, p_n$ to include the element p as well by placing p immediately to the left of p_i (if there is no such p_i, place p immmediately to the right of p_s). We thus obtain a total order $p_1, \ldots, p_r, \ldots, p_{i-1}, p, p_i, \ldots, p_s, \ldots, p_n$ of all elements of P. Denote this order by $<^1_{\text{tt}}$. Observe that the restriction of '$<^1_{\text{tt}}$' to the set P^- (we denote this restriction by '$<^1_{\text{tt}} /P^-$') is the total order $p_1, \ldots, p_r, \ldots, p_{i-1}, p_i, \ldots, p_s, \ldots, p_n$. We claim that '$<^1_{\text{tt}}$' satisfies the requirements of the Combinatorial Lemma. For this we have to show two things:

(A) '$<^1_{\text{tt}}$' extends '$<_{\text{pr}}$'. Indeed, by construction, p cannot precede in the sense of '$<_{\text{pr}}$' any element from the $p_r, \ldots, p_{i-1}$. Now suppose that for an element, say q, from the $p_i, \ldots, p_s$ we have that $q <_{\text{pr}} p$. Then by the transitivity of '$<_{\text{pr}}$' we would have that $q <_{\text{pr}} p_i$, thus contradicting the fact that '$<^1_{\text{tt}} /P^-$' extends '$<_{\text{pr}}$'. Also if such a q existed among the elements $p_{s+1}, \ldots, p_n$ then from the condition (ii) of the Combinatorial Lemma (applied not to one particular I_j but to an intersection, namely $\cap C$, of some of the I_js —it is easy to see that any intersection of any number of the I_js satisfies conditions (i) and (ii) of the Combinatorial Lemma), we would again have that $q <_{\text{pr}} p_i$, a contradiction. We get a similar contradiction if we assume that with respect to '$<_{\text{pr}}$', p precedes an element from the $p_1, \ldots, p_{r-1}$. Thus (A) is proved.

(B) Any I_j is an interval with respect to '$<^1_{\text{tt}}$'. Indeed, suppose first that I_j does not intersect the intersection $\cap C = \{p_r, \ldots, p_{i-1}, p, p_i, \ldots, p_s\}$. Then because I_j is an interval with respect to the order '$<^1_{\text{tt}} /P^-$' on $P^- = \{p_1, \ldots, p_{i-1}, p_i, \ldots, p_n\}$ and because the set $\cap C - \{p\} = \{p_r, \ldots, p_{i-1}, p_i, \ldots, p_s\}$ is not empty, we conclude that I_j must be a subset of either the set $\{p_1, \ldots, p_{r-1}\}$ or the set $\{p_{s+1}, \ldots, p_n\}$. Therefore, trivially, I_j is an interval with respect to the order '$<^1_{\text{tt}}$' on the set $P = \{p_1, \ldots, p_{i-1}, p, p_i, \ldots, p_n\}$ as well. If on the other hand, I_j intersects the intersection $\cap C$, then from the maximality of C we conclude that $I_j \in C$ and so $\{p_r, \ldots, p_{i-1}, p, p_i, \ldots, p_s\} \subseteq I_j$. However, $I_j - \{p\}$ is an interval with respect to '$<^1_{\text{tt}} /P^-$' on $P^- = \{p_1, \ldots, p_r, \ldots, p_{i-1}, p_i, \ldots, p_s, \ldots, p_n\}$. Therefore, I_j, since it contains $\{p_r, \ldots, p_{i-1}, p, p_i, \ldots, p_s\}$, is an interval with respect to '$<^1_{\text{tt}}$' on $P = \{p_1, \ldots, p_{i-1}, p, p_i, \ldots, p_n\}$ as well. This concludes the proof of (B) and of Case I.

Case II Assume that the intersection of any maximal clique is a singleton. Consider an arbitrary maximal clique, say C, and let its intersection be $\{p\}$. Because the I_js are $<^0_{\text{tt}}$-intervals of cardinality ≥ 2, it is easy to prove that there always exist two elements of the family C whose intersection is equal to $\{p\}$. Therefore, there are in C two closed $<^0_{\text{tt}}$-intervals such that the left end-point of one and the right end-point of the other are both equal to p (these end-points are taken with respect to '$<^0_{\text{tt}}$'). Now let I^{left} be the intersection of all elements in C whose right end-point (with respect to '$<^0_{\text{tt}}$') is equal to p. Similarly, let I^{right} be the intersection of all elements in C whose left end-point (with respect to '$<^0_{\text{tt}}$') is equal to p. It is easy to prove (since all the I_js have cardinality ≥ 2) that each of I^{left} and I^{right} is an $<^0_{\text{tt}}$-interval with at least two elements and that the right end-point (with respect to '$<^0_{\text{tt}}$') of I^{left} and the left end-point (with respect to '$<^0_{\text{tt}}$') of I^{right} are both equal to p. Let I denote the union $I^{\text{left}} \cup I^{\text{right}}$. Apply the induction hypothesis to the set $P^- = P - \{p\}$ with respect to the restriction of the orders '$<^0_{\text{tt}}$' and '$<_{\text{pr}}$' to P^-, and with respect to the family of intervals that comprises the sets $I_j \cap P^-, j = 1, \ldots, l$ and with respect to the family of intervals that consist of the intervals $I_j \cap P^-$, $j = 1, \ldots, l$, and $I \cap P^-$ (it is easy to check that the induction hypothesis is indeeed applicable). As in Case 1, let $p_1, \ldots, p_n$ be the total order on P^- obtained from the induction hypothesis. Then the sets $I^{\text{left}} \cap P^- = I^{\text{left}} - \{p\}$, $I^{\text{right}} \cap P^- = I^{\text{right}} - \{p\}$ and $I \cap P^- = I - \{p\}$ are nonempty intervals with respect to this inductively defined order and moreover, the first two are disjoint and their union is equal to the third. So, let $I - \{p\} = \{p_r, \ldots, p_s\}$, $I^{\text{left}} - \{p\} = \{p_r, \ldots, p_{i-1}\}$

and $I^{\text{right}} - \{p\} = \{p_i, \ldots, p_s\}$, where $r < i \leq s$ (it is possible that with respect to the order $p_1, \ldots, p_n$, the I^{left} lies not to the left but to the right of I^{right}; in this case, interchange the roles of I^{left} and I^{right}). Now define a total order '$<^1_{\text{tt}}$' on P by placing p between the elements p_{i-1} and p_i. We have to show two things:

(A) '$<^1_{\text{tt}}$' extends '$<_{\text{pr}}$'. Indeed, if there is an element, say q, among the $p_{i+1}, \ldots, p_s$ such that $q <_{\text{pr}} p$, then because $p \in I^{\text{left}}$, from condition (ii) of the hypothesis of the Combinatorial Lemma applied to an intersection (namely I^{left}) of some of the I_js, we would conclude that $q <_{\text{pr}} p_r$, a conclusion that contradicts the fact that that '$<^1_{\text{tt}} /P^-$' extends '$<_{\text{pr}}$'. All other cases are treated similarly and so (A) is proved.

(B) Any I_j is an interval with respect to '$<^1_{\text{tt}}$'. Indeed, suppose first that I_j does not contain p. Since I_j, I^{left} and I^{right} are $<^0_{\text{tt}}$-intervals, I_j cannot intersect both I^{left} and I^{right} (otherwise, it would contain p). Moreover, inductively, I_j is an interval with respect to the order $p_1, \ldots, p_r, \ldots, p_{i-1}, p_i, \ldots, p_s, \ldots, p_n$. Therefore, I_j must be a subset of either $\{p_1, \ldots, p_{i-1}\}$ or $\{p_i, \ldots, p_n\}$. In either case, it trivially follows that I_j is an interval with respect to the order $p_1, \ldots, p_{i-1}, p, p_i, \ldots, p_n$ as well. Now suppose that $p \in I_j$. As a first subcase assume that with respect to '$<^0_{\text{tt}}$', p is an end-point, say wlog the right end-point, of I_j. Then $I^{\text{left}} \subseteq I_j$ and therefore $I^{\text{left}} - \{p\}$, which is nonempty and equal to $\{p_r, \ldots, p_{i-1}\}$, is a subset of $I_j - \{p\}$. However, $I_j - \{p\}$ is an interval with respect to the order $p_1, \ldots, p_r, \ldots p_{i-1}, p_i, \ldots, p_s, \ldots, p_n$. So I_j is an interval with respect to the order $p_1, \ldots, p_{i-1}, p, p_i, \ldots, p_n$ as well. As a second and final subcase, assume that p is not an end-point (with respect to '$<^0_{\text{tt}}$') of I_j. Then $I_j - \{p\}$ must have a nonempty intersection with both $I^{\text{left}} - \{p\}$ and $I^{\text{right}} - \{p\}$. Again, it easily follows that I_j is an interval with respect to '$<^1_{\text{tt}}$'. □

We finally give the formal proof of Lemma 2.

Proof of Lemma 2 Since by hypothesis the writes-*vs*-reads matrix of ρ has the consecutive ones property, there exists a rearrangement, say R_0, of its columns such that under R_0, in every row of the matrix the 1s are consecutive (R_0 is not necessarily compatible with '$\xrightarrow{\text{xt}}$'). At first notice that '$\xrightarrow{\text{xt}}$' is a strict partial order, because it is transitive and has been assumed to be irreflexive. Now to prove the existence of a rearrangement R_1 which would keep the 1s consecutive and would also be compatible with '$\xrightarrow{\text{xt}}$', we apply the Combinatorial Lemma where we take as P the set of columns of the matrix; as $<^0_{\text{tt}}$ the ordering of the columns in R_0; as $<_{\text{pr}}$ the restriction of the partial order '$\xrightarrow{\text{xt}}$' to read operations (more accurately, to columns of the matrix as they correspond to read operations) and finally for each one of the rows w_j of the matrix (recall that the rows correspond to write operations), we take I_j to be set of columns whose entry on row w_j is 1. We then define R_1 to be the rearrangement of the columns that corresponds to the order $<^1_{\text{tt}}$. □.

Complexity Considerations By standard and easy complexity arguments, we conclude that the conditions of Theorem 4 can be checked in $O(cn^4)$ time, where c is the number of variables and n is the number of operations. Indeed, first observe that the writes-*vs*-reads matrix has at most cn nonzero entries, therefore, by the results in [5], the consecutive 1s property can be checked in $O(cn)$ time. So the only thing that remains to be computed is the time needed to define the relation '$\xrightarrow{\text{xt}}$'. Observe that for each one of the at most n^2 pairs (a, b) that may be added to '$\xrightarrow{\text{xt}}$', and for each one of the c variables, we have to look at the *uniquely* defined clans of a and b for the current variable in order to include any new pairs that should be added because of condition 4 in the definition of '$\xrightarrow{\text{xt}}$' (the other conditions of the definition of '$\xrightarrow{\text{xt}}$' do not increase the complexity more than condition 4). Since each clan contains at most n elements, checking a pair of clans takes at most n^2 steps, thus the total complexity is $O(cn^4)$. This completes the complexity analysis.

Let us now consider not the decision problem of atomicity, but the problem of actually constructing an atomic serialization of the operations, given of course that there exists one. So assume that '$\xrightarrow{\text{xt}}$' is irreflexive and that the writes-*vs*-reads matrix has the consecutive 1s property. If we are given a rearrangement R_1 of the columns of the writes-*vs*-reads matrix that satisfies the requirements of Lemma 1, i.e. that has the 1s consecutive and is compatible with '$\xrightarrow{\text{xt}}$', then we can extend the serialization $r_1, \ldots, r_k$ of the read operations defined by R_1 to an atomic serialization of all operations by recursively placing the write operations among the r_js, exactly in the same way that the sequence S_l was defined in the proof of Lemma 1. This requires $O(n^2)$ time. So the problem is reduced to the problem of constructing the rearrangement R_1. First notice that by applying any consecutive 1s algorithm to the writes-*vs* -reads matrix, we can obtain a rearrangement R_0 of the columns such that the 1s of each row are consecutive but which would not necessarily be compatible with '$\xrightarrow{\text{xt}}$'. To construct R_1 from R_0, we need an algorithmic version of the Combinatorial Lemma. Such a version can be easily obtained if we translate the inductive proof of the Combinatorial Lemma to a recursive algorithm. Notice that to find the maximal cliques of a family of intervals with respect to a total order is a complexity-wise easy problem: traverse the elements of the total order from left to right (say this order is $p_1, \ldots, p_n$) recording at each step the intervals that the current p_i belongs to; the elements of the intersection of a maximal clique are maximal sets of consecutive elements $p_r, \ldots, p_s$ all of which belong to exactly the same intervals and such that the number of these intervals, where each of the $p_r, \ldots, p_s$ belongs to, is not smaller than the number of the corresponding intervals for each of the elements p_{r-1} and p_{s+1}. Thus, we obtain an algorithm that solves not only the decision problem, but actually outputs an atomic serialization. The complexity of this algorithm is the same as the complexity of the algorithm for the decision problem.

Discussion

It is interesting to notice that closely related serializability problems in database concurrency control have been shown to be NP-complete. For example, it is proved in [18] that "view" serializability is an NP-complete problem (a view serialization is a serialization of a history of transactions on a database in such a way as the final state of the database is the same for the serial and the given history and, moreover, the the read steps of all transactions return the same values in both the serial and the given history). The NP-completeness in the case of view serializability is due to the fact that there is no requirement that the values returned by a read operation represent a snapshot of the database.

Acknowledgments

We thank Paul Spirakis. Without his encouragement we would not have attempted to continue research in the writers-readers problem. We also thank Philippas Tsigas. He supplied us with important remarks and comments during the early stages of this research. The second author would also like to thank Douglas Ierardi at the University of Southern California for his patience and support. Finally, both authors thankfully acknowledge the valuable comments of the anonymous reviewers of the previous draft.

References

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt and N. Shavit (1993): Atomic snapshots of shared memory, *Journal of the ACM* **40**, pp. 873–890. First appeared in: *Proc. of the 9th ACM Symposium on Principles of Distributed Computing, 1990*, pp. 1–14.

[2] J.H. Anderson (1990): Composite registers, *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, pp. 15–30.

[3] J.H. Anderson and M.G. Gouda (1992): A Criterion for Atomicity, *Formal Aspects of Computing: The International Journal of Formal Methods*, **4**(3).

[4] B. Awerbuch, L. Kirousis, E. Kranakis and P. Vitányi (1988): A prof technique for register atomicity (preliminary version), *Foundations of Software Technology and Theoretical Computer Science, Proceedings 8th Conference* (LNCS, Springer-Verlag), pp. 286–303.

[5] K.S. Booth and G.S. Lueker (1976): Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, *Journal of Computer and System Sciences* **13**, pp. 335–379.

[6] S. Haldar and K. Vidyasankar (1992): Elegant constructions of atomic snapshot variables, Technical Report 9204, Memorial University of Newfoundland.

[7] M. Herlihy (1991): Randomized wait-free concurrent objects, *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pp. 11–21.

[8] M.P. Herlihy and M. Wing (1990): Linearizability: a correctness condition for concurrent objects, *ACM Transactions on Programming Languages and Systems* **12**, pp. 463–492.

[9] A. Israeli and M. Li (1987): Bounded time-stamps, *Proceedings of the 28th Symposium on Foundations of Computer Science*, pp. 371–382.

[10] A. Israeli and A. Shaham (1992): Optimal multi-writer, multi-reader atomic register *Proceedings of the 11th ACM Symposium on Principles of Distributed Computing.*

[11] L.M. Kirousis, E. Kranakis and P.M.B Vitányi (1987): Atomic multireader register *Distributed Algorithms, Proceedings 2nd International Workshop* (LNCS, Springer-Verlag, 1987), pp. 278–296.

[12] L.M. Kirousis, P. Spirakis and Ph. Tsigas (1991a): Simple atomic snapshots: a solution with unbounded time-stamps, *Advances in Computing and Information, Proceedings of International Conference on Computing and Information* (LNCS, Springer-Verlag, 1991), pp. 582–587.

[13] L.M. Kirousis, P. Spirakis and Ph. Tsigas (1991b): Reading many variables in one atomic operation: solutions with linear complexity, *Distributed Algorithms, Proceedings 5th International Workshop, WDAG '91* (LNCS, Springer-Verlag, 1992), pp. 229–241. Also to appear in: *IEEE Transactions in Parallel and Distributed Systems*

[14] L.M. Kirousis, Ph. Tsigas and A.G. Veneris (1991): An atomicity criterion for composite registers, *Proceedings of the IMACS/IFAC International Symposium on Parallel and Distributed Computing in Engineering Systems* (North-Holland , 1992), pp. 31–34.

[15] L. Lamport (1978): Time, clocks and the ordering of events in a distributed system *Communications of the ACM* **21**, pp. 558–565.

[16] L. Lamport (1986): On interprocess communication, part i: basic formalism, part ii: basic algorithms, *Distributed Computing* **1**, pp. 77–101.

[17] M. Li, J. Tromp and P.M.B. Vitányi (1991): How to share concurrent wait-free variables, ITLI Prepublication Series for Computation and Complexity Theory CT-91-02, University of Amsterdam.

[18] C. Papadimitriou (1986): *The Theory of Database Concurrency Control*, Computer Science Press.

[19] G.L. Peterson and J.E. Burns (1987): Concurrent reading while writing II: the multi-writer case, *Proceedings of the 28th Symposium on Foundations of Computer Science*, pp. 383–392.

[20] A.K. Singh, J.H. Anderson and M.G. Gouda (1987): The elusive atomic register revisited, *Proceedings 6th ACM Symposium on Principles of Distributed Computing*, pp.206–221.

[21] K. Vidyasankar (1990): Concurrent reading while writing revisited, *Distributed Computing* **4**, pp. 81–85.

[22] P. Vitányi and B. Awerbuch (1986): Atomic shared register access by asynchronous hardware, *Proc. 27th IEEE Symp. on Foundations of Computer Science*, pp. 233–243.