A PROOF TECHNIQUE FOR REGISTER ATOMICITY
(Preliminary Version)

Baruch Awerbuch (1)
Lefteris M. Kirousis (2)
Evangelos Kranakis (3)
Paul M. B. Vitányi (3,4)

ABSTRACT

An implementation of a concurrent data object is wait-free if any process can complete any operation in a bounded number of steps, independently of the execution speeds of the programs. Much recent work has been done on concurrent access of shared variables by asynchronous processes. That work shows that implementing such shared variables does not require synchronization (by e.g. mutual exclusion), but can be solved in a wait-free manner. A fruitful paradigm in this context is the notion of a shared register satisfying a niceness condition called atomicity. Recent proposed solutions have led to the realization that: (1) neither the problem to be solved nor the model required were rigorously defined, (2) there was no clear insight in what constitutes a good proof of correctness in the area, and (3) the proposed protocols are so complicated that although correctness may be possible in a "platonic fashion", verifiability seems impossible to attain by human beings. A lot of controversy and allegations about constructions and proofs have arisen. Consequently, we have spent great effort to put the area on a rigorous basis. The thrust of this paper is to provide a new proof technique, and demonstrate its applicability by a non-trivial example. In other words, a new model is rigorously presented for the first time, and then a new method is given for proving register atomicity. It is then used to give a simple proof of the atomicity of the first and only direct construction of a multireader multiwriter register from atomic 1-reader 1-writer registers. (This construction was given in [8], by two of the present authors, with a completely different proof.)

(1) Department of Mathematics and Laboratory for Computer Science, MIT, Cambridge MA 02139, USA
(2) University of Patras, Department of Mathematics, Patras, Greece
(3) Centrum voor Wiskunde en Informatica, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
(4) Faculteit Wiskunde en Informatica, Universiteit van Amsterdam, The Netherlands

1. Introduction

Concurrency control of asynchronous processes is often realized by actively serializing concurrent actions, using synchronization primitives like mutual exclusion, semaphores, and locking. Thus, although it *seems* that the actions are executed concurrently, in the system they are *actually* executed serially in some order. It has been pointed out in [3] that to implement such primitives we first need interprocess communication through a shared memory unit, which we shall call a *register*, even if the processors communicate by message passing. This suggests that the problem of simultaneous memory access needs to be solved without recourse to synchronisation primitives. It is desired that such a solution involves *no waiting* by one operator for another one. Thus we kill two birds with one stone, since it is the waiting involved in synchronization methods to control the communication between asynchronous participants, which may make such solutions unacceptable. Note, that asynchrony need not be due solely to hardware, but can also be caused by multiple users on the various machines. The problem of providing general wait-free asynchronous communication interfaces becomes more acute, as more and more hardware from different technologies, scale and speed continue to be connected in computer networks and other complexes. The purpose of the present investigation is to examine the feasibility of such general interfaces. In particular, we analyse the problem of how to implement a shared register which can be read by different asynchronous processors (the readers) and be written by different asynchronous processors (the writers) in a truly concurrent fashion. That is, without any restrictions to prevent simultaneous access and making no assumptions, either about the relative durations of the reads and writes, or about the actual timing of the lower level constituent operation executions.

More precisely, we are given some registers with certain restrictions on their mode of operation, e.g. that only a certain number of operators are allowed to access each one of them. We are asked to construct a more powerful (*compound*) register without some of the original restrictions, while retaining some of the positive characteristics of the subregisters, e.g. their serializable mode of operation (otherwise called *atomicity*). These compound registers will comprise a set of registers (i.e. *subregisters*) and an operation execution on the compound register will consist of a sequence of operation executions on the subregisters that follow a given *protocol*. All we require from the constructed protocol is that it guarantees the *existence* of some total (i.e., linear) order in which the operation executions on the compound register could have taken place (external consistency). This order, in some sense, represents the succession these operations *seemingly* follow. Of course, for such a total order to be meaningful, it must satisfy certain additional requirements. For example, there should be no second write placed by this order between a read and the write it reads (internal consistency). If we assume that there is a global (i.e. referring to all registers) time-reference system (otherwise, *a global clock*), and if all subactions of an operation execution on the compound register precede in time all the subactions of a second operation execution on the compound register, then this order must place the second operation execution after the first one. In general, we have a relation on the operation executions which is naturally imposed by the problem (e.g., an acyclic relation that tells if an operation execution can have an influence on another), and we

desire the existence of a total order that extends this relation, but without violating the above restrictions. If this is possible for each scenario of operation executions of a proposed register, then the register is atomic.

In the next section we present the model rigourously, and give two general atomicity criteria that are suitable for proving register atomicity (the second criterion is a simple variation of the first, but is more suitable for 1-writer registers). Our work is in the spirit of the general 'causality' model proposed by Lamport [3], and we have no need to assume the existence of a global clock. It considerably extends Lamport's work, notably so by treating multiwriter registers in an order setting for the first time. Lamport avoids this issue, and is explicit that his framework only covers single writer registers. Moreover, we focus sharper on concurrent register access, as opposed to concurrent operation executions in general. We also investigate how the assumption of global time affects these criteria, by proving a rather general shrinking function theorem. In Section 3 we prove the atomicity of a multiwriter, multireader compound register directly constructed from atomic 1-writer, 1-reader subregisters. This is the first atomic multiwriter register, and was introduced by two of us in [8]. Using unbounded tags, it incorporates the essence of the problem area in a simple and comprehensible algorithm. At this time of writing, it still is the only published direct implementation from atomic 1-reader 1-writer registers. The atomicity proof given here is based solely on causality considerations.

Previous work in this area is due to Bloom, Lamport, Peterson, and Vitányi and Awerbuch (see [1], [3], [6], [8]). More recent work can be found in PODC87 and FOCS87.(*) Generally, the algorithms and proofs, especially of the ambitious bounded tag algorithms in [8] and in the related papers in PODC87 and FOCS87, defy comprehension even by other workers in the field. Reference [8] contains the fairly simple unbounded tag algorithm we present here, and a complicated bounded tag algorithm for a subcase of the general question, together with a proof method. The bounded tag algorithm contains an error (see Errata, FOCS87). A 'patched up' version of this algorithm was presented by Peterson and Burns in FOCS87 [7], but was recently found to be erroneous as well by Russell Schaefer. The reader, when consulting these references, will find that inventing the 'right' formalism and level of rigour is of major importance in this complicated area. In this paper we have succeeded developing *simple* formal criteria and proof methods, and demonstrate their usefulness by proving a major but simple protocol correct in a *convincing* manner.

2. The Model

A **proto-register** is an abstract data type, capable of holding **values** out of a given **domain** of values. Initially the proto-register is empty. The operations that can be performed on the proto-register are writes and reads. A *write* of a value puts that value in the proto-register. A *read* reports a value from the domain.

(*) PODC87 stands for "Proceeding of 6th ACM Symposium on Principles of Distributed Computing" held in Vancouver, Canaca, 1987, and FOCS87 stands for "28th Annual IEEE Symposium on Foundations of Computer Scince" held in New York, USA, 1987.

We assume that such values have an identity apart from a value. That is, values written by different write operation executions may have the same value, e.g. 0, but they are not identical. The **identity** $id(v)$ of a value v written by a write w is defined by $id(v)=w$. (Thus, $v=v'$ and $id(v)\neq id(v')$ may be both true.) If a read operation execution reports a value v, then either $id(v)=w$ for a particular write w or else $id(v)$ is undefined. If a read r reports v with $id(v)=w$ for some write w, then we say that r reports the value written by w. The proto-register can be implemented by a **multiset** over a given domain. That is, an unordered list of elements, where the same element can occur more than once. A proto-register has associated with it a finite set of processors called the **writers** and a finite set of processors called the **readers**. A processor can be both a reader and a writer. A write can only be performed by a writer, while a read can only be performed by a reader.

A **sequential** register is a proto-register where all operations are executed in sequence, and an execution of a read operation reports the value written by the execution of the last write operation that precedes it. A sequential register can be implemented by a **linear list**. Originally, the list is empty. A write adds an element to the end of the list, and a read reports the element at the end of the list.

We address the problems arising from true concurrency, where we allow simultaneous operation executions by different processors. However, simultaneous operation executions by the same processor are excluded. Informally, we aim at a specification of a general **concurrent** register, register for short, which corresponds as closely as possible to that of the sequential register. In the atomic register defined below, the operations may be **actually** executed concurrently, yet it will **seem** as if they were executed in sequence. For read operations to report a value which was written by a write operation to the register, there must be causal relations between the operations. We define an 'apparent' precedence relation ($\rightarrow$) on the set of operation executions, which captures the crucial aspect of the causal relations between operation executions to the same register.

Remark. Lamport [3] defines two precedence relations $\rightarrow$ and $--\rightarrow$, the semantics of which are intended to be problem independent. If a "precedes" relation on the subactions of a,b is defined, then $a --\rightarrow b$ means "some subaction of a precedes some subaction of b", and $a \rightarrow b$ means "each subaction of a precedes each subaction of b." In the context of shared register access there is always an intended way for the actions to interact, which ensures correctness of the algorithm. Therefore, it is advantageous to reflect this essential causal relation between the actions of a particular algorithm by a single made-to-measure precedence relation. This relation is our $\rightarrow$, not to be confused with Lamport's $\rightarrow$, and will have an algorithm dependent semantics.

To define various degrees of niceness conditions on a register with simultaneous operation executions, we need some formal definitions first. We use 'action' as synonym for 'operation execution'.

A **run** $\rho = (A, \rightarrow, \pi)$ consists of the following:

(R1) A finite or countably infinite set A of **read** and **write** actions. If R is the set of read actions and W is the set of write actions, which were actually performed during the

course of the run, then $A = W \cup R$ and $W \cap R = \emptyset$.

(R2) A **reading mapping** which is a partial function $\pi : R \to W$.

(R3) An irreflexive partial order $\to$ on the set A of actions. We call $\to$ a **precedence relation**. If $a \to b$ then we say a **precedes** b. To initialize the run, there is an **initial** write that precedes all other actions. We moreover require that, for each $a \in A$, there are only finitely many $b \in A$ such that $\neg(a \to b)$. Informally, this means that a run begins at some point in time, rather than extending in the infinite past [3], and that an action cannot be infinitely long or infinitely small in duration.

Intuitively, $a \to b$ will imply that, in the aspect we deem important, a may influence b, but b cannot influence a. Two actions a, b are called **concurrent** if $\neg(a \to b$ or $b \to a)$. I.e., if they are incomparable in the relation $\to$. If w is a write and r is a read , then w **directly precedes** r, **if** $w \to r$ **and there is no write** w', **such that** $w \to w' \to r$.

Irreflexive orders.

All **orders** in this paper are irreflexive. For convenience, "total order" and "partial order" will henceforth mean "irreflexive total order" and "irreflexive partial order," respectively.

A run $\rho = (A, \to, \pi)$ can now be classified into the following categories according to how well it behaves under concurrent operations. The definitions below closely follow the presentation of Lamport [3]. The 'normal' run is a new category we found advantageous to introduce.

1. **(safe)** For each read r, that has no concurrent writes, $\pi(r)$ is defined, and directly precedes r.
2. **(normal)** For each read r, $\pi(r)$ is defined, and $\pi(r)$ either precedes r or is concurrent with r.
3. **(regular)** For each read r, $\pi(r)$ is defined, and $\pi(r)$ directly precedes r or is concurrent with r. (Hence a regular run is both safe and normal.)
4. **(atomic)** A run is atomic if it is normal and there is a total order $\Rightarrow$, which we call an **atomic precedence** relation, on the set A of actions, as follows.
   (i) if $a \to b$ then $a \Rightarrow b$ **(external consistency)**, and
   (ii) for each read r, $\pi(r)$ is the write directly $\Rightarrow$-preceding r **(internal consistency)**.

   We say that $\Rightarrow$ **atomically extends** the precedence relation $\to$.

Without proof we state the hierarchy involved. Every atomic run is regular, but not every regular run is atomic. By definition, regular runs are exactly the ones which are both safe and normal. There are runs which are safe but not normal, and there are runs which are normal but not safe.

A run is a possible set of operation executions by a register, a possible 'history'. We now tie up the notion of a register and the notion of a run. Intuitively, a register is a deterministic 'black box' that reports a value in response to a read query. We can view

the function of this black box as associating a reading mapping with a given pair $(A,\rightarrow)$. Since $(A,\rightarrow)$ is a high-level description, it is actually an equivalence class of different finer grained descriptions. These differences may give rise to different responses to the same read query. Therefore, the register associates a *set* of reading mappings with each $(A,\rightarrow)$. Let Π be the set of all its possible reading mappings. Formally, a **register mapping** $REG: \{(A,\rightarrow)\}\rightarrow 2^{\Pi}$ is a total mapping, that associates a nonempty set of reading mappings π with each pair $(A,\rightarrow)$ satisfying (R1), (R2) and (R3). With each register we associate a register mapping. We assume that each processor actually executes operations to the register serially. This assumption is embodied in requirement (R4) below. If K is a register and REG_K is its associated register mapping, then a run $\rho=(A,\rightarrow,\pi)$ of K satisfies

(R4) if a and b are different actions by the same processor then either $a\rightarrow b$ or $b\rightarrow a$, and this total $\rightarrow$-order on the actions by the same processor is identical with the serial order in which a processor executes its actions in A;

(R5) $\pi \in REG_K(A,\rightarrow)$; and

(R6) if a read r returns a value v and $id(v)=w$, then $\pi(r)=w$.

A register is **atomic** (respectively **regular, normal, safe**) if each of its runs is atomic (respectively regular, normal, safe). Obviously, the atomic register is the ideal register; the operations may be concurrent, yet they seem to be executed in a serial fashion, extending the given precedence relation (external consistency) and consistent with the reading function (internal consistency).(*) Given a particular implementation of a data structure, and having selected the particular precedence relation $\rightarrow$ we wish to employ, it is often simple to check whether it is a safe, a normal, a regular register or none of these. However, proving atomicity using the given definition, or its 'shrinking' variant which we will meet below, turns out to be a difficult matter. Therefore, in the next section we propose simple criteria which are necessary and sufficient for atomicity. In a later section we show how to use these atomicity criteria for verifying that a proposed construction implements an atomic register.

2.1. Atomicity Criteria

For a proposed data structure K to be an atomic register, it suffices to prove that each of its runs, as defined in (R1) through (R6), is atomic. Let $\rho = (A,\rightarrow,\pi)$ be a normal run of K, so π is total. We divide the set of actions A into equivalence classes induced by π. Each such equivalence class, called a clan, is associated with a write. The **clan** associated with a write w is the set $[w] = \{w\}\cup\{r \in R : \pi(r) = w\}$. For any two writes w, w'

(*) The notion of register atomicity is closely related to what is called '(strict) serializability' in conventional concurrency control, in particular in the context of databases with concurrent 'transactions'. See e.g. [5]. Concurrent transactions are usually called atomic if they are both serializable and **recoverable**. Recoverability means that each transaction appears all-or-nothing: either it executes to completion (in which case we say that it **commits**) or it cannot influence other transactions (in which case we say that it **aborts**). Recoverability is a problem only in the presence of failures. We assume that registers are failure-free, so we do not consider recoverability.

define $[w] \rightarrow^{\pi} [w']$ if and only if $w \neq w'$ and there exist actions $a \in [w]$ and $a' \in [w']$ such that $a \rightarrow a'$. Note that $\rightarrow^{\pi}$ is not necessarily acyclic. The following theorem is basic for proving the atomicity of runs.

Theorem 2.1. (Atomicity Criterion)

Let $\rho = (A, \rightarrow, \pi)$ be a run. The following statements are equivalent:

(1) ρ is atomic.

(2) ρ is normal and $\rightarrow^{\pi}$ is acyclic.

Proof.

(1) implies (2). Let ρ be atomic. By definition, atomicity implies normality, which shows the first part of (2). To show the second part of (2), let $\Rightarrow$ be a total order that atomically extends $\rightarrow$. I.e., for each read r, we have

(i) $\pi(r) \Rightarrow r$, and

(ii) there is no write w with $\pi(r) \Rightarrow w \Rightarrow r$.

We prove that $\rightarrow^{\pi}$ is extendible to a total order, which implies acyclicity of $\rightarrow^{\pi}$. It is enough to show that for any two writes w, w', if $[w] \rightarrow^{\pi} [w']$ then $w \Rightarrow w'$.

Since $\Rightarrow$ is a total order, the negation of $w \Rightarrow w'$ is equivalent to $w' \Rightarrow w$. Therefore, we only need to show that for any two writes w, w', if $w' \Rightarrow w$ then $\neg([w] \rightarrow^{\pi} [w'])$. Thus, suppose $w' \Rightarrow w$. Exhaustive analysis of all cases shows that then the combination of (i) and (ii) implies $w' \Rightarrow r' \Rightarrow w \Rightarrow r$, for all reads $r \in [w]$ and $r' \in [w']$. Hence, there are no $a \in [w]$ and $a' \in [w']$ such that $a \rightarrow a'$. Therefore, $\neg([w] \rightarrow^{\pi} [w'])$.

(2) implies (1). Assume (2) holds. It is clear that the transitive closure of $\rightarrow^{\pi}$ is a partial order, which in turn can be extended to a total order $\Rightarrow^{\pi}$. Since ρ is normal, for each read $r \in [w]$, we have $\neg(r \rightarrow w)$. Hence, there is a total order $\Rightarrow_{[w]}$ on each $[w]$ atomically extending $\rightarrow$ and such that $w \Rightarrow_{[w]} r$, for each read $r \in [w]$. Define a unique relation $\Rightarrow$ on the set A as follows. For all $a, a' \in A$, $a \Rightarrow a'$ if and only if either

(i) $a, a' \in [w]$ and $a \Rightarrow_{[w]} a'$, or

(ii) $a \in [w]$, $a' \in [w']$, and $[w] \Rightarrow^{\pi} [w']$.

Clearly, $\Rightarrow$ is a total order atomically extending $\rightarrow$. It follows that ρ is atomic. •

The second atomicity theorem refers to registers with only one writer. In this case, for each pair of different writes $w, w' \in A$, either $w \rightarrow w'$ or $w' \rightarrow w$, by (R4). The theorem is similar to a corresponding theorem in Lamport [3].

Theorem 2.2. (1-writer Atomicity Criterion)

Assume that K is a register with only one writer. Then for each run $\rho = (A, \rightarrow, \pi)$ of K the following statements are equivalent:

(1) ρ is atomic.

(2) ρ is regular and π is weakly monotonic (i.e., if $r \rightarrow r'$, then either $\pi(r) \rightarrow \pi(r')$ or $\pi(r) = \pi(r')$).

Proof.

(1) implies (2). Let ρ be atomic. Atomicity implies regularity. Therefore we only need to prove weak monotonicity of π.

Let $r, r' \in A$ be different reads with $r \rightarrow r'$. Since there is only one writer, we have by (R4) that either $\pi(r) \rightarrow \pi(r')$ or $\pi(r) = \pi(r')$ or $\pi(r') \rightarrow \pi(r)$. Atomically extend $\rightarrow$ to a total order $\Rightarrow$, as in the definition of an atomic run. Exhaustive case analysis shows that, by the properties of $\Rightarrow$, either $\pi(r) \Rightarrow r \Rightarrow \pi(r') \Rightarrow r'$, or $\pi(r) = \pi(r')$.

(2) implies (1). Let ρ be regular and π be weakly monotonic. By Theorem 2.1, if we prove that $\rightarrow^{\pi}$ is acyclic, then we are done. Assume to the contrary, there is a cycle

$$[w] \rightarrow^{\pi} [w'] \rightarrow^{\pi} \cdots \rightarrow^{\pi} [w] .$$

Since $[w] \rightarrow^{\pi} [w']$, there are $a \in [w]$ and $a' \in [w']$, $w \neq w'$, such that $a \rightarrow a'$. If both a and a' are writes then $w \rightarrow w'$. If a, a' are both reads, then by weak monotonicity of π, we have $w \rightarrow w'$. If a is a read and $a' = w'$, then $a \rightarrow w' \rightarrow w$ $(= \pi(a))$ contradicts normality of ρ. Therefore $w \rightarrow w'$ by (R4). If $a = w$ and a' is a read, then $w' \rightarrow w \rightarrow a'$ $(\pi(a') = w')$ contradicts safety of ρ, and therefore $w \rightarrow w'$ by (R4) again. Hence, $[w] \rightarrow^{\pi} [w']$ implies $w \rightarrow w'$. Since this argument holds for all pairs of adjacent clans in the cycle, we obtain a cycle $w \rightarrow w' \rightarrow \cdots \rightarrow w$. This contradicts that $\rightarrow$ is a partial order. •

2.2. Compound Register

The most obvious approach to constructing a register is to build it from simpler ones. The existence of such a simpler register is either postulated, or it is constructed from still simpler registers. More precisely, a **compound register** consists of a finite number of registers, called **subregisters**. The set of readers and writers of the compound register is the union of the set of readers and writers of the subregisters. The subregisters are allowed to hold a value out of a given domain. We can distinguish essentially two cases. In one case the subregisters are simpler than the compound register in that their domain of values is smaller than the value domain of the compound register. Then the construction for the compound register **distributes** the value to be stored piecemeal over the subregisters. E.g., a positive integer n can be distributed in $\log n$ bits over $\log n$ boolean subregisters. In the other case the set of readers and writers associated with the compound register is larger than the set of readers and writers associated with each subregister. Then the construction for the compound register **replicates** the value to be stored as versions in several subregisters. A reader has to determine the 'latest' version among the versions it obtains from different subregisters. To make this possible, extra information such as a 'timestamp' is attached to each version. As a result, the value domain of each subregister has to be larger than the value domain of the compound register. The construction of a compound register in this paper is of the latter type. To express their complexity we use the following cost measures. Let V be the value domain of the compound register, $v = |V|$, and let there be n readers and m writers associated with the compound register. Let $S, T: V \times \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$ be total cost functions, with $\mathbf{N}$ the set of nonnegative integers. Let the value domain of each subregister of the compound register be (isomorphically) contained in $TAG \times V$, with $|TAG| = S(v, n, m)$, the number of elements in TAG. Then the **space** complexity of the compound register is $\log S(v, n, m)$. The processors execute read or write actions on the compound register, independently of each other but following a protocol. Let each read or write action by a given processor on the

compound register consist of at most $T(v,n,m)$ read and/or write actions on the subregisters. Then the **time** complexity of the compound register is $T(v,n,m)$. An action on the compound register is considered to be a higher-level operation execution of the same nature as its subactions. Thus, with each run of the compound register is associated a run of each subregister which constitutes the compound register. This means that we associate with each subregister a set of subactions related by a precedence relation. Sets of subactions associated with different subregisters are disjoint. We assume that a processor actually executes all its subactions in serial order. The disjoint precedence relations of the subactions on respective subregisters are related by the order in which each processor executes its subactions. For the compound register we define a transitive precedence relation ($-\gg$) on the set of all subactions involved, as follows.

Let K be a compound register comprising subregisters $K_1, \ldots, K_n$. Let $\rho=(A,\rightarrow,\pi)$ be a run of K and let $\rho_i=(A_i,\rightarrow_i,\pi_i)$ be the associated run of subregister K_i, $1\leq i\leq n$. The **precedence** relation $-\gg$ on the set $\bigcup_{i=1}^{n} A_i$, is defined as the minimal transitive relation that extends all precedence relations $\rightarrow_i$, $1\leq i\leq n$, such that

(R7) if α and β are different subactions by the same processor, then either $\alpha-\gg\beta$ or $\beta-\gg\alpha$, but not both, and this total $-\gg$-order on the subactions by the same processor is identical with the actual serial order in which the processor executes these subactions; and

(R8) if $a,b\in A$ and for each subaction α of a and each subaction β of b holds $\alpha-\gg\beta$, then $a\rightarrow b$.

Lemma 2.3.

$-\gg$ is a partial order.

Proof.

Clearly, (R7) precludes $-\gg$-cycles containing two subactions by the same processor. Therefore, since the sets of subactions on the same subregisters are disjoint, any $-\gg$-cycle contains only subactions on the same subregister. But these subactions are partially ordered, which contradicts such a $-\gg$-cycle. •

Finally, we need to express the 'registerhood' of the compound by suitably restricting the choice of $\rightarrow$. That this is necessary can be seen from the following example. Let K be a compound register consisting of subregisters K_1,K_2. Let p be a writer associated with subregister K_1, and let q be a reader associated with subregister K_2, $p\neq q$. Then there is no way that q can read what p has written. Yet runs of K can satisfy (R1) through (R8) and even be atomic. For example, atomicity of K_1,K_2 implies atomicity of K. Such anomalies are due to the fact that we have not yet required the existence of causal relations between actions by different processors. There must be some causal relation between a write and a read, since otherwise a reader cannot report what a writer wrote. There must be some causal relation between two writes, because otherwise a writer cannot replace the value in the register by the value it wants to write. However, it is not necessary to have a causal relation between two reads; this is because neither do reads have to change the value contained by the register, nor do they need to report what the another read wrote. The following condition expresses these requirements on the

compound register in terms of subregisters. Assuming the general setting above:

(R9) if $a, b \in A$ are not both reads, then there are subactions α of a and β of b, α, β are not both subreads, and some i $(1 \leq i \leq n)$, such that $\alpha, \beta \in A_i$. (α and β act on the same subregister K_i.)

It follows that a choice of $\rightarrow$ satisfying (R9) is 'proper' if the choices of the $\rightarrow_i$'s are 'proper.' This can be argued as follows. Assume that the ultimate subsub..subregister is atomic. If a, b are not both reads, then there are subactions α of a and β of b, not both reads, which act on the same subregister, and so on. At the atomic subsub..subregister level the subsub..subactions involved have an apparent total order. Choose this as the precedence relation. For convenience, let the K_i's be the basic atomic subregisters, so the $\rightarrow_i$'s are total orders. Then either $\alpha \rightarrow_i \beta$ or $\beta \rightarrow_i \alpha$, but not both, by (R7). Suppose $\alpha \rightarrow_i \beta$. If $c, d \in A$, $c \rightarrow a$ and $b \rightarrow d$, then by (R8) we have $c \rightarrow d$. Suppose $\beta \rightarrow_i \alpha$. If $c, d \in A$, $a \rightarrow c$ and $d \rightarrow b$, then by (R8) we have $d \rightarrow c$. Using the precedence relations at the previous level, we induce in this fashion a 'coarse' precedence relation at each next higher level compound register. Our choice of $\rightarrow$ is constrained to be an extension of this coarse precedence relation. That is, (R7) through (R9) restrict the freedom of our choice of $\rightarrow$ appropriately, by ultimately reducing the constraints on our choice of precedence $\rightarrow$ to precedence at the elemental level.

2.3. Naming of Registers

Unfortunately, the naming conventions for types of registers are inconsistent. For a 1-writer register, the operator who writes can simply remember the value it wrote last. Therefore, the name '1-writer, 1-reader' register is used for a register that can be read by both writer and reader [3]. By analogy, we use '1-writer, $(n-1)$-reader' register for a register that can be written by one writer and read by $n-1$ readers that cannot write. The writer can always read as above. However, in an m-writer register, with $m > 1$, while a writer can remember what it wrote last, this value can have been overwritten by a later write of another writer. Hence, here we might as well have writers that cannot read (in addition to readers that cannot write). We will, however, only consider registers where the writers can also read. For us, an 'm-writer, n-reader' register, $m > 1$, designates a register that can be written by m processors, and read by n processors including the m writers $(n \geq m)$.

2.4. New Proof Technique

In the present paper we propose a new proof technique for proving the atomicity of compound registers. In summary, our approach consists of the following method:

1. Find an appropriate partial order $\rightarrow$ between the high level reads and writes defined in terms of the assumed partial order between the lower level reads and writes. Induce the $\rightarrow^{\pi}$ relation on the set of clans defined by the reading mapping.
2. Find a way to totally order the writes, using the intuition which makes you believe the protocol works correctly. Use this total order to prove that $\rightarrow^{\pi}$ is acyclic.

3. Example: Multiwriter Register

The matrix register is a compound n-writer, n-reader register constructed as a matrix of atomic 1-writer, 1-reader subregisters. The domain of values of the subregisters is the cartesian product of the domain of values of the compound register with the nonnegative integers. This is the first atomic multiwriter register [8], and at the time of writing still is the only direct construction from atomic 1-reader 1-writer subregisters. It may well be a register of practical importance, because of its simplicity, elegance and low complexity (cf. below). We prove correctness by application of the atomicity Theorem (Theorem 2.1).

Architecture.

Let $p_1,\ldots,p_n$ be n processors and let K be an $n \times n$ matrix register consisting of n^2 atomic, 1-reader, 1-writer registers $K_{i,j}$, $i,j = 1,\ldots,n$. Each p_i is a writer of (i.e., is connected to the write terminal of) each $K_{i,j}$. Each p_i is also a reader (i.e., is connected to the read terminal) of each $K_{j,i}$. Let V be the domain of values of the compound register. Then $\mathbf{N} \times \{1,\ldots,n\} \times V$, with $\mathbf{N}$ the nonnegative integers, is the domain of values of each subregister. A **tag** is a pair (k,i), where k is a nonnegative integer and $i \in \{1,\ldots,n\}$. We say that each subregister can hold a tag, next to a value from the domain V of the compound register. All subregisters are initialized with tag (0,1) and value 0. Moreover, each run of the compound register starts with a write action, which precedes all other actions, as required by (R3). The architecture is depicted in Figure 1.

Protocol.

The register K obeys the following protocol.

p_i writes the value v:

1. for all $j = 1,\ldots,n$ read $K_{j,i}$ (i.e., read the ith column);
2. determine the lexicographically largest tag $(k_{\max},m)$;
3. set own tag to $(k_{\max}+1, i)$;
4. for all $j = 1,\ldots,n$ write on $K_{i,j}$ (i.e., write to the ith row) the new tag, as well as the value v.

p_i reads:

1. for all $j = 1,\ldots,n$ read $K_{j,i}$ (i.e., read the ith column);
2. determine the lexicographically largest tag $(k_{\max},m)$ and let v_m be the value contained in a register with such a tag;
3. set own tag to $(k_{\max},m)$;
4. for all $j = 1,\ldots,n$ write to $K_{i,j}$ (i.e., write to the ith row) the new tag, as well as the value v_m, which was determined in 2. (Also, report v_m.)

Each action a by processor p_i consists of a set of subreads $R(a,1,i),\ldots,R(a,n,i)$ followed by a set of subwrites $W(a,i,1),\ldots,W(a,i,n)$, where the last two indices i,j

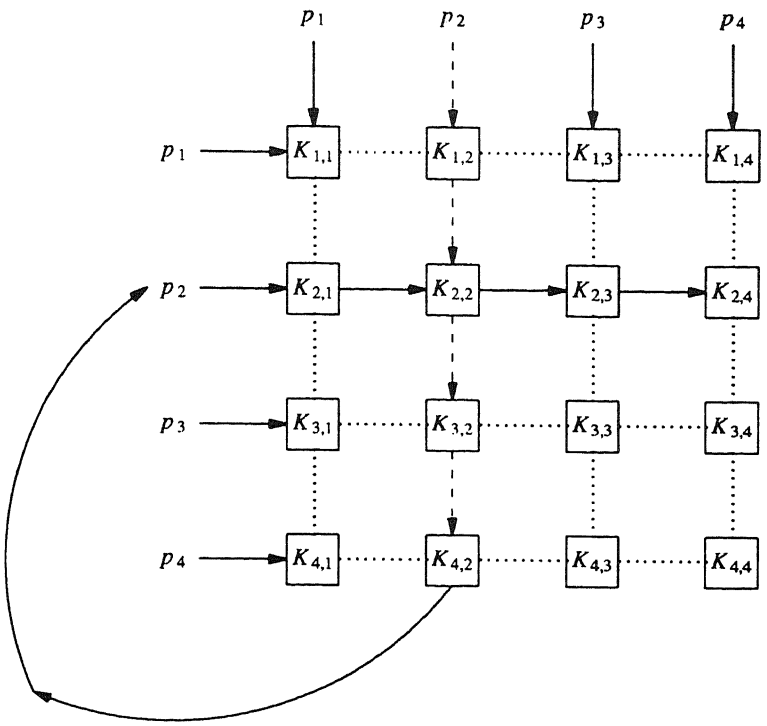

Figure 1: An action by processor p_2 in the 4-reader, 4-writer, matrix register.

indicate the subregister $K_{i,j}$ on which the subaction took place. The order in which these subreads and subwrites take place is arbitrary, but for the fact that each subread precedes each subwrite. Each subregister $K_{i,j}$ $(1\leq i,j\leq n)$ of K is atomic. Let $\rho=(A,\rightarrow,\pi)$ be a run of K. Let A^i be the subset of actions in A that are executed by p_i $(1\leq i\leq n)$. Define, for all $1\leq i,j\leq n$, $\rho_{i,j}=(A_{i,j},\rightarrow_{i,j},\pi_{i,j})$, the run of $K_{i,j}$ associated with ρ, where

$$A_{i,j}=R_{i,j}\cup W_{i,j},$$

$$R_{i,j}=\{R(a,i,j): a\in A^j\},$$

$$W_{i,j}=\{W(a,i,j): a\in A^i\};$$

$R_{i,j}$ is the set of subreads, and $W_{i,j}$ is the set of subwrites on subregister $K_{i,j}$. Since $K_{i,j}$ is atomic, there is an atomic extension $\Rightarrow_{i,j}$ of $\rightarrow_{i,j}$, for all $1\leq i,j\leq n$. This atomic extension is a total order on the subactions executed on the subregister concerned. Moreover, if a subread reads a subwrite (on a subregister), then there is no other subwrite placed between them by this order. The orders on the disjoint sets of subactions associated with each subregister are related by the orders on the disjoint sets of subactions by each processor. Let $-\!\!\gg$ be the minimal transitive relation on the subactions in $\bigcup_{i,j=1}^{n}A_{i,j}$ extending the $\Rightarrow_{i,j}$'s and satisfying (R7). By (R7), the subactions by the same processor p are totally ordered by $-\!\!\gg$. This order is the serial execution order of the subactions by

p. In particular, $-\gg$ must satisfy:

$$R(a,i,j) -\gg W(a,j,k), \tag{3.1}$$

for all $a \in A^j$ and all $1 \le i,j,k \le n$. We now define a precedence relation $\rightarrow$ on A. For any two actions a and b on the compound register K, by p_i and p_j, respectively, let $\rightarrow$ be the transitive closure of $\rightarrow'$:

$$a \rightarrow' b \text{ iff } W(a,i,j) -\gg R(b,i,j). \tag{3.2}$$

Clearly, this satisfies (R8) and (R9).

Lemma 3.1.

$\rightarrow$ is a partial order on A.

Proof.

Existence of a $\rightarrow$-cycle containing $a \in A^j$, implies $W(a,j,k) -\gg R(a,i,j)$, for some k,i $(1 \le k,i \le n)$. This contradicts (3.1), since $-\gg$ is a partial order by Lemma 2.3. •

Remark. If $-\gg$ extends the original partial orders $\rightarrow_{i,j}$, instead of the apparent total orders $\Rightarrow_{i,j}$, then Lemma 3.1 still holds for the $\rightarrow$ resulting from (3.2). This will be useful in the proof of Theorem 3.5.

The following theorem is the main result of this section.

Theorem 3.2.

The matrix register K is an atomic, n-writer, n-reader compound register, which is implemented with n^2 atomic, 1-writer, 1-reader registers.

Proof.

Let $\rho = (A, \rightarrow, \pi)$ be a run of K. Examine the write protocol. For a write $w \in A$, let $v(w)$ be the value written by w to the compound register, and, if $t(w)$ denotes the tag determined in step 3 of w, let $(t(w), v(w))$ be the value written to the subregisters in step 4 of w's execution. For a read $r \in A$, let $v(r)$ be the value reported by r from the compound register, and, if $t(r)$ is the tag associated with $v(r)$, let $(t(r), v(r))$ be the value written to the subregisters in step 4 of its execution. The pair $(t(r), v(r))$ is selected in step 3 of the read protocol.

Claim. For each read r, there is a write w, such that

$$t(r) = t(w) \;\&\; id(v(r)) = id(v(w)) \; (=w). \tag{3.3}$$

If $id(v(r)) = w$ then $\pi(r) = w$. Hence, π is total.

Proof of Claim. The subregisters are initialized with tag (0,1), and there is a write preceding all other actions, by (R3). Hence, there is an $a \in A$, such that $(t(r), v(r)) = (t(a), v(a))$. If a is a write then we are done, else a is a read and we repeat the argument. If $r \rightarrow a$, then, by (3.2) and atomicity of the subregisters, r can not read the value written by a to the subregister involved. There are only finitely many a, such that $\neg(r \rightarrow a)$, $\rightarrow$ is a partial order, and there is an initial write preceding all other actions, by (R3). Hence, we need only finitely many repetititions of the argument before we find a write w such that (3.3) holds. If $id(v(r)) = w$ then $\pi(r) = w$ by (R6). Since

this holds for each read r, π is total. This proves the Claim.

Let $<_{lx}$ be the irreflexive **lexicographic** order on pairs of integers. If $a,b \in A$ such that $a \rightarrow b$ (therefore $a \neq b$), then it follows by (3.2) and the choosing of the new tag in step 3 of the write and read protocols, that:

$$t(a) \leq_{lx} t(b) \quad (t(a) <_{lx} t(b) \text{ if } b \in W). \tag{3.4}$$

The prove atomicity, by Theorem 2.1, we only need to prove that ρ is normal and that there is a total order extending the $\rightarrow^{\pi}$ relation among the clans. Intuitively, we proceed by *first* choosing a plausible total order on the set of writes, and *next* showing that the corresponding total order on the set of clans extends $\rightarrow^{\pi}$. In the matrix register, the obvious total order on the set of writes is the lexicographical order of the associated tags. Proceeding this way, the conclusion of the theorem follows from Theorem 2.1 and by the following lemma.

Lemma 3.3.

(1) ρ *is normal, and*

(2) if $[w] \rightarrow^{\pi} [w']$, *then* $t(w) <_{lx} t(w')$. *In particular,* $\rightarrow^{\pi}$ *is acyclic.*

Proof.

(1). By the Claim above, π is a total function. Let $\pi(r)=w$ (i.e., $r \in [w]$). Then, by (3.3), $t(r) = t(w)$. However, if $r \rightarrow w$, then by (3.4) we have $t(w) >_{lx} t(r)$, which is a contradiction. Hence, $\neg(r \rightarrow w)$, i.e., ρ is normal.

(2). Let $[w] \rightarrow^{\pi} [w']$. By definition of $\rightarrow^{\pi}$, there exist actions $a \in [w]$ and $b \in [w']$ such that $a \rightarrow b$.

Suppose $b = w'$. Then by (3.3) and (3.4) it follows that $t(w') >_{lx} t(w)$, which is as claimed.

Suppose that $a = w$ and b is a read. By (3.3) and (3.4), $t(w) \leq_{lx} t(b) = t(w')$. If w, w' are writes by different processors, then their tags have different processor numbers; if they are writes by the same processor then, since $w \neq w'$, one of them $\rightarrow$-precedes the other. Therefore, by (3.4), they must have different tags. In both cases, $t(w) \neq t(w')$, which is as claimed.

Suppose both a, b are reads. Then $t(a) = t(w) \leq_{lx} t(b) = t(w')$, by (3.3) and (3.4). The proof of $t(w) \neq t(w')$ is now exactly as before. This proves the lemma. Hence the proof of theorem 3.2 is complete. •

3.1. Complexity and Optimality

The time complexity of the matrix register is $2n$ (or rather $2n-2$, as follows from Theorem 3.4 below) which seems to be as low as it can possibly be. The space complexity of the matrix register is unbounded. In theory this is pretty bad. In practice, however, this solution uses far less space than many solutions which theoretically do better. For instance, in [8] a solution has been proposed where the space complexity of the compound register is $5n^2 \log n$. However, we can assume that a system executes only a limited number of actions on the compound register in its total lifetime. If we set a generous

bound of at most 2^{50} such actions, the matrix solution is superior in terms of space complexity, with respect to the mentioned bounded space solution, for any number $n \geq 3$ of associated processors. Thus the matrix register has effectively a lower space complexity than comparable solutions with bounded space complexity, even for solutions which solve only subproblems of the one addressed by the matrix solution. An exception is the Bloom register in [1] (with only two writers), which both effectively and theoretically cannot be improved in space and time complexity.

Another complexity criterion is the number of subregisters of a certain type used in the compound register. Leaving out the subregisters on the main diagonal, which are redundant, the matrix solution is optimal in the number of 1-writer, 1-reader subregisters used.

Theorem 3.4. (Optimality)

The implementation of a compound safe n-writer, n-reader register from 1-writer, 1-reader subregisters, requires at least $n(n-1)$ such subregisters (atomic or not). Register K, minus the subregisters on the main diagonal, is such an optimal implementation.

Proof.

Suppose we have implemented a safe compound n-writer, n-reader register R, with associated processors $p_1, \ldots, p_n$, from 1-writer, 1-reader subregisters. For each ordered pair of processors (p_i, p_j), $1 \leq i, j \leq n$ and $i \neq j$, we can consider a run $(\{w, r\}, \rightarrow, \pi)$ of R, consisting solely of two nonoverlapping operation executions: a write w by p_i, followed by a read r by p_j. Since R is safe, $\pi(r) = w$. Since $i \neq j$, there must be a subregister $R_{i,j}$, such that p_i is the associated writer and p_j is the associated reader. There are $n(n-1)$ different ordered pairs (p_i, p_j), $i \neq j$. In each such ordered pair the first element is a writer and the second element is a reader. No subregister $R_{i,j}$ can be associated with more than one such (writer, reader) pair, since the subregisters have only one associated writer and one associated reader other than the writer. Hence, there must also be $n(n-1)$ different subregisters $R_{i,j}$ in the compound register R. This is exactly achieved by the presented matrix register K, noting that the subregisters on the main diagonal are superfluous. I.e., p_i can remember what it wrote last in $K_{i,i}$. •

4. Global Time, Intervals and Shrinking

In [1], [6], [7], [8], atomicity is related to the assumption of a global time reference frame (also called global clock). We show that the theory as developed here is more general. In particular, a register is atomic in global time if and only if it is atomic for a particular choice of the $\rightarrow$ precedence relation. This precedence relation turns out to be the interval order induced by the time intervals representing the actions.

An important aspect of atomic runs is the following property. Although their actions have a duration on a global time scale, and such durations may overlap, each action may be considered to take place instantaneously, i.e., as if it happened completely at a particular time instant. If all of these time instants are distinct, then the apparent time instants of the actions orders the actions totally. This relates the order approach to

atomicity with the global time approach.

Time is represented by the set of real numbers **R**, ordered as usual. Assume that every action a is **represented** by an open time interval $(s(a), f(a))$, $s(a)<f(a)$, within the bounds of which the action is supposed to have taken place. $s(a)$ (respectively, $f(a)$) is a real number called the **starting** (respectively, **finishing**) time of the action a.

Remark. To exclude some technical difficulties, there is usually an assumption that $s(a)\neq s(b)$, $s(a)\neq f(b)$ and $f(a)\neq f(b)$, for any two distinct actions $a,b\in A$, and $s(a)\neq f(a)$ for each action $a\in A$. The fact that we should be allowed to assume that no two starting or finishing times are equal, is justified by appeal to the sensibility of natural law [3]. "No physical meaningful result could depend on upon completely accurate knowledge of these times. (It makes no physical sense to specify starting and finishing times of an operation execution down to the fraction of a micropicosecond.)" By excluding the starting and finishing times from the duration associated with action a, to obtain the desired effect in the mathematical framework, we may come closer to the spirit of physics. Thus, we choose to represent durations of actions as open intervals.

Define the precedence relation $\rightarrow$, as the natural relation $a\rightarrow b$ iff $f(a)\leq s(b)$. A relation which is so induced by a set of intervals of the real line **R**, satisfies the axioms of a special type of partial order called interval order. Formally, an **interval order** on a set A is an irreflexive relation $\rightarrow$ that satisfies

$$a\rightarrow b \;\&\; c\rightarrow d \text{ implies } a\rightarrow d \text{ or } c\rightarrow b, \text{ for all } a,b,c,d\in A \text{ (see [2])}. \qquad (4.1)$$

Every interval order is a partial order, and hence the previously developed theory applies. Since not every (irreflexive) partial order is an interval order, the global time approach requires more from the precedence relation ($\rightarrow$) than the general approach in (R3). We now proceed with the definitions of this more conventional global time approach to atomicity. The relation between the two approaches is analysed in Theorem 4.1.

A **shrinking function** on the set of actions of a run $\rho=(A,\rightarrow,\pi)$, with $\rightarrow$ the interval order induced by the set of intervals $\{(s(a),f(a))\subseteq\mathbf{R}: a\in A\}$, is a one-to-one function σ that associates with each action a of the run a time instant (i.e., a real number) $\sigma(a)$ such that:

(S1) $\sigma(a)$ belongs to the interval $(s(a),f(a))$ of a.

A shrinking function gives a possible serialization of the actions. Condition (S1) enforces external consistency of the serialization. In the order approach, external consistency follows from the fact that the serialisation extends $\rightarrow$. Define the precedence relation $\rightarrow_\sigma$, induced by σ, as $a\rightarrow_\sigma b$ iff $\sigma(a)<\sigma(b)$. Obviously, $\rightarrow_\sigma$ is a total order on A. Then (S1) implies that $\rightarrow_\sigma$ extends $\rightarrow$. A shrinking function σ is consistent with the reading mapping π if

(S2) $(A,\rightarrow_\sigma,\pi)$ is atomic.

A run ρ is **shrinking** atomic if there is a shrinking function σ such that (S1) and (S2) are satisfied.

Theorem 4.1. (Shrinking Function Theorem)

Let $\rho=(A,\rightarrow,\pi)$ be a run, and let $\rightarrow$ be the interval order induced by a representation of open (time) intervals of the actions in A. The following statements are equivalent:

(1) ρ is atomic, and

(2) ρ is shrinking atomic.

Proof.

(1) implies (2). Suppose (1) holds. Let $\Rightarrow$ be a total order which atomically extends $\rightarrow$. Define $Q(a)=\{b:\neg(a\rightarrow b)\ \&\ \neg(b\Rightarrow a)\}$. Note that, by (R3), $Q(a)$ is finite, and that $Q(a)$ is nonempty since $a\in Q(a)$. Define, by induction on a, $\sigma(a)$ to be a real number such that:

(i) if $b\Rightarrow a$ then $\sigma(a)>\sigma(b)$,

(ii) $\sigma(a)>s(a)$, and

(iii) $\sigma(a)<\mu$, with $\mu=\min\{f(b):b\in Q(a)\}$.

Note that (ii) and (iii) imply (S1), and (i) implies (S2). Induction is possible if:

(a) if $b\rightarrow a$ then $\sigma(b)<\mu$, and

(b) $s(a)<\mu$.

Let $b_{\min}\in Q(a)$ be an action such that $\mu=f(b_{\min})$.

Ad (a). Assume $b\rightarrow a$. If $b\rightarrow b_{\min}$ then $\sigma(b)<f(b)\leq s(b_{\min})<f(b_{\min})=\mu$.
If $\neg(b\rightarrow b_{\min})\ \&\ \neg(b_{\min}\rightarrow b)$ then, since $b\rightarrow a$, we have $\neg(b_{\min}\Rightarrow b)$. Therefore, $b_{\min}\in Q(b)$. Then, $\sigma(b)<\min\{f(c):c\in Q(b)\}\leq f(b_{\min})=\mu$.
If $b_{\min}\rightarrow b$ then $b_{\min}\rightarrow b\rightarrow a$, contradicting $b_{\min}\in Q(a)$.

Ad (b). Since $\neg(a\rightarrow b_{\min})\ \&\ \neg(b_{\min}\Rightarrow a)$ we have $s(a)<f(b_{\min})=\mu$.

(2) implies (1). Assume (2). Since $\rightarrow_\sigma$ is a total order extending $\rightarrow$ and satisfying (S2), atomicity of ρ is immediate. •

Corollary.

The matrix register is shrinking atomic.

Proof sketch.

The argument goes as follows. Assume global time. The interval representations of the subactions induce the $\rightarrow_{i,j}$ precedence relations on the subregisters. Each such relation is therefore an interval order. The intervals associated with the subactions of each processor are linearly ordered (do not overlap) by definition. Since each subregister $K_{i,j}$ is atomic, each run $\rho_{i,j}=(A_{i,j},\rightarrow_{i,j},\pi_{i,j})$ has a shrinking function $\sigma_{i,j}$ such that $(A_{i,j},\rightarrow_{\sigma_{i,j}},\pi_{i,j})$ is shrinking atomic, by Theorem 4.1. Since the associated intervals are open, we can always choose the $\sigma_{i,j}$'s such that $\sigma'=\bigcup_{i,j=1}^{n}\sigma_{i,j}$ is one-to-one. Define $-\gg$ as the total order of the real images of the subactions under σ', i.e., $-\gg$ agrees with the usual total order $<$ on the reals. Then $-\gg$ satisfies (R7) and (3.1). Define $a\rightarrow'' b$ iff $\sigma'(\alpha)<\sigma'(\beta)$ for all subactions α of a and β of b. Then $\rightarrow''$ is an interval order. This satisfies (R8) and (R9), and $\rightarrow$ is a refinement of $\rightarrow''$. The proof of Theorem 3.2 goes

through exactly as before, with interval order $\rightarrow''$ instead of $\rightarrow$, which implies that register K is shrinking atomic by the Shrinking Function theorem (Theorem 4.1). •

5. Conclusion

To recapitulate the main result of this paper we propose a method of proving atomicity of shared registers in an order setting. It seems to us that it can be applied in many cases where we have to prove atomicity (e.g. we have used the methods presented in the present paper to prove the atomicity of the 2-writer register given in [1]). In outline:

1. Find an appropriate partial order $\rightarrow$ between the high level reads and writes defined in terms of the assumed partial order between the lower level reads and writes. Induce the $\rightarrow^{\pi}$ relation on the set of clans defined by the reading mapping.
2. Find a way to totally order the writes, using the intuition which makes you believe the protocol works correctly. Use this total order to prove that $\rightarrow^{\pi}$ is acyclic.

Acknowledgement

Conversations with Bard Bloom, Leslie Lamport, Arjen Lenstra and Nancy Lynch are gratefully acknowledged. Lambert Meertens' comments had a profound influence on this paper.

References

[1] Bloom, B., *Constructing Two-writer Atomic Registers,* Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing, Vancouver, Canada, 1987.

[2] Fishburn, P.C., *Interval Orders and Interval Graphs,* Wiley, 1985.

[3] Lamport, L., *On Interprocess Communication, Part I: Basic Formalism, Part II: Algorithms,* Distributed Computing, vol. 1, pp. 77-101, 1986.

[4] Lamport, L., *The mutual exclusion problem, part I - A theory of interprocess communication,* Journal ACM, vol. 33, pp.313-326, 1986.

[5] Papadimitriou, C., *The serializability of concurrent database updates,* Journal ACM, vol. 26, pp. 631-653, 1979.

[6] Peterson, G. L., *Concurrent Reading While Writing,* ACM Transactions on Programming Languages and Systems, Vol. 5, No. 1, Jan. 1983, pp. 46-55.

[7] Peterson, G.L. and J.E. Burns, *Concurrent Reading While Writing II, the Multi-writer Case,* Proceedings 28th IEEE Symposium on Foundations of Computer Science, New York, USA, 1987.

[8] Vitanyi, P. M. B., and Awerbuch, B., *Atomic Shared Register Access by Asynchronous Hardware,* Proceedings 27th IEEE Symposium on Foundations of Computer Science, 1986, 233-243. (Errata, in Proceedings 28th IEEE Symposium on Foundations of Computer Science, New York, USA, 1987.)