The Peterson-Burns Multi-Writer, Multi-Reader

Atomic Register Algorithm

by

Russel W. Schaffer

Submitted to the Department of

Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements

for the Degree of

Bachelor of Science in Computer Science and Engineering

at the Massachusetts Institute of Technology

May 1988

Author _____
Department of Electrical Engineering and Computer Science

Certified by _____
Nancy Lynch
Thesis Supervisor

Accepted by _____
Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

The Peterson-Burns Multi-Writer, Multi-Reader Atomic Register Algorithm

by

Russel W. Schaffer

Submitted to the
Department of Electrical Engineering and Computer Science

May 16, 1988

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering

ABSTRACT

The construction of a multi-writer, multi-reader atomic register has been addressed by several papers by different authors. As a result of the difficult nature of the problem, it is not easy to grasp the intuition behind some of these algorithms, and the proofs of correctness provided are sometimes not as rigorous as one might desire. There is, however, one paper by Bard Bloom that addresses a less general problem and is both intuitively appealing and completely rigorous. It is the purpose of this paper to provide both an intuitive feel for and a rigorous proof of correctness of the more general algorithm developed by Peterson and Burns; Bloom's paper is used as a model for this paper. In the process of developing the proof of correctness, two problems were found with the algorithm. This paper thus presents these counterexamples and a proof of correctness for the modified algorithm.

Thesis Supervisor: Nancy Lynch
Title: Professor of Computer Science and Engineering

Part I
Introduction and Intuition

1 Introduction

The problem of constructing a multi-writer, multi-reader atomic register was first introduced in [LL]. It has, at this point, been addressed by several papers by different authors [BB],[IL],[LV],[PB]. As a result of the difficult nature of the the problem, however, most of these papers are rather hard to understand; it is not generally easy to grasp the intuition behind some of the algorithms, and the proofs of correctness provided are sometimes not as rigorous as one would desire for a problem of this difficulty. There is, however, one paper on the subject that distinguishes itself as both intuitively appealing and completely rigorous; that paper presents a construction for the specific case of a two-writer, multi-reader atomic register [BB]. It is the purpose of this paper to attempt to provide both an intuitive feel for and a rigorous proof of correctness of the more general algorithm presented in [PB]; [BB] is used as a model for this paper. Consequently, many of the facts proved in this paper are the same as or resemble those proved in [BB] or [PB]. The terminology and notation of these papers has been largely retained in the interest of consistency.

In the process of developing a proof of correctness for the algorithm from [PB], several problems were found in the code that could cause the algorithm to work improperly. Communication with one of the authors of [PB] concerning the first of these problems confirmed that the problem did exist and that the counterexample presented to the code exploited a case that was inadequately considered in the [PB] proof of correctness. Indeed, the immediately obvious fixes to the code proved to be futile, as a final couterexample communicated by Burns demonstrates, and a more modest correction had to be made to the algorithm for work on the proof of correctness to continue. Another unrelated problem arose at a later date. This problem has an obvious solution. This paper thus presents both counterexamples to the correctness of the original algorithm and a proof of correctness of a modified version of the algorithm.

2 The Model and the Problem

This paper presents the algorithm from [PB] within the framework of the I/O automaton model developed in [LT]. The following loose and cursory description of that model is sufficient for one to comprehend the remainder of the paper.

An I/O automaton may be thought of as a set of states, a set of actions, and a transition function mapping the product of the sets of states and actions to the set of states. Thus when the automaton is in one state and a particular action occurs, the transition function specifies the new state that the automaton is to enter.

An execution of an I/O automaton may be thought of as an alternating sequence of states and actions of the automaton; each action marks a transition of the automaton from the state that preceeds the action in the sequence to the one that follows the action in the sequence.

The actions of an I/O automaton are subdivided into three disjoint sets, the internal actions, output actions, and input actions. An automaton controls the states from which its internal and output actions may occur, however, it must be capable of accepting an input action while in any state. Since automata may be formally composed, we may thus think of two automata as communicating along a channel if an output action of one automaton corresponds, in a composition, to an input action of the second automaton.

The problem of constructing an m-writer n-reader atomic register will thus be seen as one of constructing an I/O automaton with the following actions and properties:

1. The automaton should have m channels along which the input actions $Start(W)$, for writes W, may be accepted. This implies that the symbol $Start(W)$ subsumes m distinct input actions which we will differentiate by explicit reference to their respective writers. To each $Start(W)$ action corresponds a unique $Finish(W)$ action.

   Each $Start(W)$ action thus represents a request by its particular writer to begin the write W to the m-writer n-reader atomic register. When such an action occurs, the writer that received it begins execution of its writer's protocol; when execution of the protocol terminates, the automaton executes the $Finish(W)$ action that corresponds to the $Write(W)$ action that initiated the write.

2. Similarly, the automaton should have n channels along which the n input actions $Start(R)$ may be accepted for reads R. Again, to each of these n distinct actions there corresponds an output action $Finish(R)$. As was the case with writers, each $Start(R)$ action initiates execution of the reader's protocol by a reader, and is followed, after termination of execution of the protocol, by an action $Finish(R)$.

3. Given any execution e of the I/O automaton, it should be possible, for every read R to insert into e an internal action $Atomic(R)$ between $Start(R)$ and $Finish(R)$, and for every write W to insert into e an internal action $Atomic(W)$ between $Start(W)$ and $Finish(W)$, to yield a new execution e' of the automaton with the following property. For every read R in e', if W_R is the write whose value was returned by R then $Atomic(W_R)$ must be the last *Atomic* action for any write before $Atomic(R)$ in e'.

   Note that it is assumed that along any given channel, the initiators of the *Start* actions will wait until a corresponding *Finish* action has been received along that channel before performing another *Start* action along that channel; if this condition is not met, the behavior of the automaton will remain unspecified.

3 The Composition Automaton

Let us begin consideration of the Peterson-Burns algorithm for constructing an m-writer n-reader atomic register from 1-writer $n+m$-reader atomic registers by presenting the configuration of automata that would implement the multi-writer multi-reader atomic register. With each writer is associated a 1-writer $m+n$-reader atomic register that may be written by that writer alone, but which may be read by any of the writers or readers. This is illustrated by figure 1.

In the figure 1, the circles represent distinct I/O automata, and the lines represent channels between them. The heavy lines represent write channels, while the lighter lines represent read channels.

Each *Register i* represents a single writer, $m+n$-reader atomic register automaton that has the following actions associated with writes W and reads R:

$Start(W)$ This input action serves as a request on the register's write channel to initiate the write W of some value to the register.

$Atomic(W)$ This internal action denotes the point at which the write W may be thought to have occured atomically.

$Finish(W)$ This output action serves as a signal on the write channel that the write W has completed.

$Start(R)$ This input action serves as a request on some read channel to initiate the read R of the current value of the register.

$Atomic(R)$ This internal action denotes the point at which the read R may be thought to have occured atomically.

$Finish(R)$ This output action serves as a signal, on the read channel along which $Start(R)$ originated, that the read R has completed.

Each *Writer i* denotes an I/O automaton executing the Peterson-Burns writer's protocol. It has the following actions of interest associated with each write W that it performs:

$Start(W)$ This input action serves as a request on the i'th write channel to initiate the write W of some value to the m-writer n-reader atomic register. Note that this is thus an input action of importance to the composition register as well.

$1Scan(W)_j$ This action is actually an internal action of the m-writer n-reader atomic register that corresponds to the $Atomic(R)$ action of a particular read R of *Register j*. More on its significance later. Note that in this case, as with all other subscripted actions we define, we are actually defining one such action for each j, $1 \leq j \leq m$.

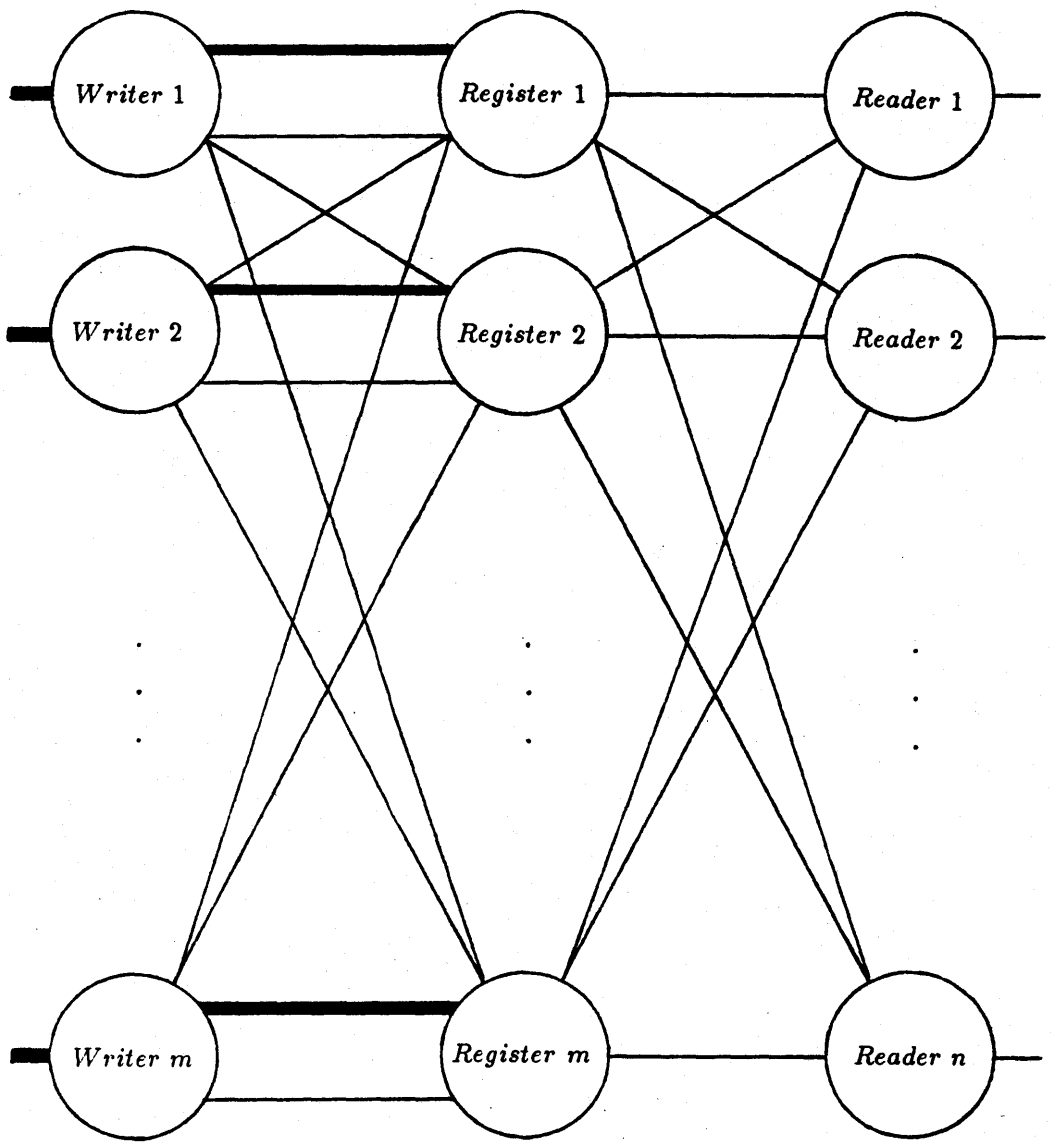

Figure 1: The composition automaton.

$PWrite(W)_j$ Like $1Scan(W)_j$, this is an internal action of the m-writer n-reader atomic register; it corresponds to the $Atomic(W')$ action of a particular write W' to *Register* i. More will be said about it later.

$2Scan(W)_j$ Analogous to $1Scan(W)_j$.

$Scan(W)$ An internal action inserted in an execution immediately following the action $2Scan(W)_m$. The proof of correctness will show that the values returned by the $xScan(W)_i$ are identical to those contained in all of the writers' registers at the point $Scan(W)$.

$PScan(W)_j$ Analogous in definition, though not in meaning, to $1Scan(W)_j$.

$3Scan(W)_j$ Analogous to $1Scan(W)_j$.

$Write(W)$ This is analogous in definition, but not in meaning, to $PWrite(W)_j$. It is at this point that the value, $Value(W)$, being written by the write W is actually written to *Register* i.

$Finish(W)$ This output action serves as a signal on the i'th write channel that the write W has completed. Note that this is thus an output action for the composition automaton as well.

Note that for a write during which all these actions are performed, these actions occur in the order in which they are presented above.

Finally, each *Reader* i denotes an I/O automaton executing the Peterson-Burns reader's protocol. We will refer to the following of its actions associated with a read R that it performs:

$Start(R)$ This input action serves as a request on the i'th read channel to initiate the read R by reader i of the value of the m-writer n-reader atomic register. Note that this is thus an input action important to the composition register as well.

$1Scan(R)_j$ This action is actually an internal action of the m-writer n-reader atomic register that corresponds to the $Atomic(R')$ action of a particular read R' of *Register* j performed during the read R. More on its significance later.

$2Scan(R)_j$ Analogous to $1Scan(R)_j$.

$Finish(R)$ This output action serves as a signal on the i'th read channel that the read R has completed. Note that this is thus an output action for the composition automaton as well.

Note that for a read during which all these actions are performed, these actions occur in the order in which they are presented above.

It is the task of our proof of correctness to associate with each read R and each write W the internal actions $Atomic(R)$ and $Atomic(W)$ respectively. These are analogous to the $Atomic(R')$ and $Atomic(W')$ actions associated with reads R' and writes W' of the 1-writer, $m+n$-reader automata.

4 The Version Numbers

So far we have established the composition automaton that executes the Peterson-Burns algorithm. We will now present a bit of intuition to explain how the algorithm should work. Note that this is anything but a proof of correctness.

When a reader automaton receives a request to begin a read of the value in the m-writer n-reader atomic register implemented by the composition automaton described earlier, it must somehow figure out which writer contains a value that is the correct one to return. To aid in this process, each writer maintains a set of "version numbers" which are visible to the readers and on the basis of which a current value may be selected. The information maintained by each writer i in its register is as follows:

$VN[i,j]$ Every time writer i performs a write that does not time out (we will discuss what that means later) to the m-writer n-reader atomic register, a new value of $VN[i,j]$ is written into writer i's register for every writer j. The rules for choosing the new $VN[i,j]$ will be discussed later.

$PVN[i,j]$ Even though writer i changes its $VN[i,j]$ every time it performs a write that does not time out, the old value of $VN[i,j]$ does not immediately disappear; whenever the value of $VN[i,j]$ changes, its old value is rewritten by writer i into its register as the value $PVN[i,j]$.

$OVN[i,j]$ In the process of performing a write W, writer i reads the version numbers contained in the other writers' registers and writes them into its own register; the value read for $VN[j,i]$ is written by writer i into its register as $OVN[i,j]$. These values essentially record the global state of the VN's at the time of the write W; they thus serve as a sort of timestamp to communicate the relative recency of the value, $Value[i]$ in register i.

$Value[i]$ At the same time that it writes the $VN[i,j]$, $PVN[i,j]$, and $OVN[i,j]$, writer i also writes to its register the value, $Value(W)$, that it is in the process of writing to the m-writer n-reader atomic register. This value is written by writer i into its register as $Value[i]$.

$PreOVN[i,j]$ This value is used only by writers and is not visible to readers. It contains either the current value of $OVN[i,j]$, or a value of $OVN[i,j]$ that writer i is planning to write but has not yet written.

It is sometimes difficult to keep all of these different indexed variables straight; a partial aid to remembering them is provided by noting that the first index of a variable is always the index of the writer in whose 1-writer $m+n$-reader register the variable resides. The $VN[i,j]$ reside in the register of writer i and are thus written exclusively by writer i; similarly for the other indexed variables.

Another important point to remember is that the first four variables, the $VN[i,j]$, $PVN[i,j]$, $OVN[i,j]$, and $Value[i]$, are written to the register of writer i only once

during any write W by writer i. Since the atomic action of this write to writer i's register corresponds to the action $Write(W)$, we conclude that the values of these variables remain constant between such actions. The values of the $PreOVN[i,j]$ change at other times.

5 The Reader's Protocol

The importance of these variables to reads is that by examining the relative values of the VN, PVN, and OVN, a reader should be able to determine to a large extent which writers wrote most recently. Consequently, a reader is capable of determining which of the $Value[i]$ is most likely to be the correct one to return. The following facts are useful in this respect:

1. If at some point $OVN[i,j] = VN[j,i]$, then as of that point, the most recent write by writer i is somewhat more recent than the most recent write by writer j. This is so for the following reason: when writer i was selecting the value of $VN[j,i]$ to write as $OVN[i,j]$ during its last write, it chose the value $VN[j,i]$ written by the most recent write by writer j; this implies that the most recent write by writer i was still deciding what to write after the point where the most recent write by writer j had already written. Loosely speaking, we say that writer i "sees" the version number $VN[j,i]$ that was written by the most recent write by writer j. This means that if writer i "sees" writer j's version number, then the last write by writer i is relatively more recent than that of writer j.

   Note that this is a "fact" only inasmuch as the correct operation of the algorithm depends upon its truth; the second counterexample exploits a breakdown of this fact.

2. If writer i "sees" neither the VN nor the PVN of writer j, that is if $OVN[i,j] \neq VN[j,i]$ and $OVN[i,j] \neq PVN[j,i]$ at some point, then as of that point, the most recent write by writer i is considerably less recent than that by writer j. This is so because writer j must have written at least twice since the most recent write by writer i was selecting the value of $VN[j,i]$ it would write as $OVN[i,j]$. This would imply that the value contained in $Value[i]$ is particularly archaic; in general, a read should avoid returning such a value.

3. At no point does any writer ever "see" its own version number; that is, at all points, $OVN[i,i] \neq VN[i,i]$. At the same time, however, every writer always "sees" its own PVN; at all points $OVN[i,i] = PVN[i,i]$.

Of these three facts, the first is by far the most important. Indeed, it captures the essence of the purpose of the version numbers. It is on the basis of this fact that we make the following definition. At a given point for a given writer i, we will define $VNS(i)$ to be:

$$VNS(i) = \{j | 1 \leq j \leq m, OVN[i,j] = VN[j,i]\}.$$

It is an important fact about the VNS that, if at any point, $VNS(i) \neq VNS(j)$, then either $VNS(i)$ is a proper subset of $VNS(j)$ or $VNS(j)$ is a proper subset of $VNS(i)$ at that point. This means that at each point there will be some writer k for which $VNS(i) \subset VNS(k)$ for all writers i. The first fact from above implies that if $VNS(i)$ is a proper subset of $VNS(k)$ for some writer i, that is if writer i "sees" the version numbers of fewer writers than does writer k, then $Value[k]$ should be treated as being more recent than $Value[i]$. Since set inequality implies set inclusion, we conclude that $|VNS(i)|$ is a valid measure of the relative recency of the last write of $Value[i]$.

Unfortunately, $|VNS(i)|$ is not an adequate measure of recency to provide a basis for a read to uniquely pick a "correct" value to return. It is possible to have two separate writers i and j, $i \neq j$, that wrote at more or less the same time resulting in $VNS(i) = VNS(j)$. Thus an additional measure of the recency of a write is needed. To this end we will employ the second fact from above and define, for a given point and a given writer i, the value $N(i)$ at that point to be:

$$N(i) = \begin{cases} 1 & \text{if for all writers } j, \text{ either} \\ & OVN[i,j] = VN[j,i] \text{ or } OVN[i,j] = PVN[j,i] \\ 0 & \text{otherwise.} \end{cases}$$

By the second fact from above, $Value[i]$ for a writer i for which $N(i) = 1$ should be considered to be more recent than $Value[j]$ for a writer j for which $N(j) = 0$. It would be quite desirable if the two measures of recency that we have just defined, $|VNS(i)|$ and $N(i)$, did not contradict each other; that is, if $|VNS(i)| > |VNS(j)|$ then $N(i) \geq N(j)$. If these two measures did not contradict each other, then the sum $N(i)+|VNS(i)|$ would serve as a better measure of recency than $|VNS(i)|$ alone. The problem that leads to the second counterexample, however, may be exploited to show that these measures are not always consistent with each other. In the modified algorithm, these values do have consistent meanings.

Unfortunately, even when $|VNS(i)|$ and $N(i)$ are consistent measures of recency, $|VNS(i)|+N(i)$ is still not an adequate measure of recency of $Value[i]$ to provide readers with a criterion for picking a unique value, $Value[i]$, to return. It is again possible for $|VNS(i)| + N(i) = |VNS(j)| + N(j)$ even though $i \neq j$. Fortunately, $|VNS(i)| + N(i)$ is a strong enough measure of recency that we can make the following definition, for a given point, of F at that point: if M is the maximum value of $|VNS(i)| + N(i)$ for any writer i, then let F be the largest numbered writer for which $|VNS(F)|+N(F) = M$. It is clear that at any point, the value of F is unique. It falls upon our proof of correctness to show that $Value[F]$ is always a legitimate value for a read to return.

So far we have explained how a reader goes about choosing a correct value to return based on observed values of the VN, PVN, and OVN. What we have not done is to state how a reader goes about reading a consistent set of such values. If a reader were simply to scan the writers' registers in succession, starting with an atomic read of all the values in writer 1's register and finishing with an atomic read of the values in writer m's register, then if we were to compute F on the basis of the values observed,

$Value[F]$ need not be a correct value to return. It is entirely possible that the writers could write, as the scan is taking place, in such a manner that the values observed by the reader's scan are entirely unrealistic causing the reader to return the value of a write that is very much out of date.

This is clearly undesirable behavior. So we ask if a reader would get a consistent set of values if it were to scan the values of the writers' registers twice, starting with an atomic read of the values in writer 1's register through an atomic read of writer m's register followed by another atomic read of writer 1's register and so on through a final atomic read of the values in writer m's register. If we were to require that the values $VN[i,j]$ observed by the first scan be identical with the values $VN[i,j]$ observed by the second scan for all writers i and j, would the second scan yield a set of values from which we could determine F such that $Value[F]$ is a valid value to return? This is the approach adopted by the code in [PB]. This approach does not work as we will see later; this is the basis for the first counterexample. Indeed, even if one were to require that not only the VN's but the PVN's and the OVN's as well remain constant across the two scans, then the second scan still does not return a set of values for which $Value[F]$ is necessarily a correct value to return. The modified version of the algorithm that we will prove correct incorporates a suggestion by Burns that a reader require that all of the VN's, OVN's, and PVN's remain constant across *three* consecutive scans of the writers' registers.

Note that if these two consecutive, identical scans are performed as part of a read R, then the action $1Scan(R)_i$ corresponds to the "atomic" action of the read of the values in writer i's register performed during the first of the two scans. Similarly for the actions $2Scan(R)_i$.

There is still one question about the way the read protocol determines the value of F that remains unresolved. It is entirely possible that a reader could perform an infinite sequence of scans and never see two consecutive scans that are identical. To solve this problem, readers keep track of the writers whose values they have seen change between scans. If, in the course of a read R, it is observed that a writer i has changed its values two times, then because writes by a single writer are not permitted to overlap in time, the write W_2 that caused the second change of value must have started after the end of the write W_1 that caused the first change of value. Since changing the values visible to readers is the last step in the writer's protocol, we conclude that essentially the entire write W_2 was performed after the start of the read R but before the scan that observed the second change in the values in writer i's register. This means that to return the value, $Value[i]$, written by the write W_2 is to return a legitimate value for the read R; the point at which we can think of the write W_2 as having occured atomically will necessarily be contained within the bounds of R so if we think of R as having occured immediately after that point, we see that it is valid if $Value(R) = Value(W_2)$. If a reader observes that a writer i has changed its value twice, then it will take this course of action, returning the value of $Value[i]$ observed after the second change; reads that return a value determined in such a way are said to have "timed out."

By the pigeon hole principle, it is necessary that after $m+2$ consecutive scans of the registers, either two consecutive scans have returned the same values for all of the writers, or some writer has been seen to change its values at least twice. Thus, by the time at most $m+2$ scans have been completed as part of a read, that read has either timed out, or terminated normally having completed two consecutive scans that return the same values. Similarly, after $2m+3$ consecutive scans, either three consecutive scans have been observed to be identical or some register has been seen to change value twice.

In summary, the [PB] reader's protocol operates as follows:

1. A reader performing a read first scans the writers' registers attempting to make two consecutive scans that return the same values of $VN[i,j]$ for all writers i and j. By the end of at most $m+2$ scans, either two such scans will have been observed, or the read will have timed out returning a value written by a writer whose values have been observed to change twice. If two consecutive scans return the same values of the $VN[i,j]$ then the values observed by the second scan are used in the next step to determine the value to return.

2. On the basis of the values read in the first step, the values of $|VNS(i)|$, of $N(i)$, and finally of F are computed. The value of $Value[F]$ seen during the second of the two consecutive, identical scans from the first step is then returned.

   Note that the code in [PB] actually returns the value $Value[F]$ seen during the first scan; this is in plain contradiction with the correctness arguments given in that paper and has been corrected in the code in this paper.

This concludes our discussion of how readers choose the values they are to return.

6 The Writer's Protocol

We have discussed a reader's choice of a value to return based on the existence of several variables maintained by the writers. We have yet to demonstrate how these variables are maintained. We will do so now.

Just as a reader must first read the values in all of the writers' registers to determine what value to return, so too a writer must first read all of the writers' registers to determine what to write. Writers read a consistent set of values in a manner almost identical with that by which readers obtain a consistent set of values (although the reason why the method works is somewhat different in the two cases). As before, a writer obtains a consistent set of values for the VN, PVN, and OVN by making scans of the writers' registers. This time, if across three consecutive scans, none of the VN, PVN, or OVN are seen to change, then the writer may assume that the values read by the last of the three scans represent a consistent state of the world on the basis of which the writer may complete its write. Again, if these three scans are performed as part of a write W, then the action $1Scan(W)_i$ corresponds to the "atomic" action of

the read of the values in writer i's register performed during the first of the three scans; similarly for the actions $2Scan(W)_i$ and $3Scan(W)_i$. It is an important fact that at the point where the action $Scan(W)$ was placed, the values in all of the writers' registers equal those read by the three scans, thus we may think of the three scans as having occured atomically at the point $Scan(W)$. We will consequently refer only to $Scan(W)$, the scan point of W, throughout the remainder of this paper, and ignore the separate scans where possible.

The $PreOVN$ are read somewhat differently. The $PreOVN$ are not read as part of the scans of the writers' registers. Rather, after three consecutive, identical scans of the writers' registers have been performed as above, say as part of a write W by writer i, the $PreOVN[j,i]$ are read sequentially. Each is read only once. This is supposed to be adequate to obtain a meaningful set of values for the $PreOVN[j,i]$; it turns out, however that it is not. This problem is corrected by performing the reads of the $PreOVN[i,j]$ between the second and third of the three consecutive identical scans. It is to the "atomic" read action of the read of a $PreOVN[j,i]$ that the action $PScan(W)_j$ corresponds.

Assuming that a writer i has succeeded at reading a consistent set of values for the $VN[j,k]$, $PVN[j,k]$, $OVN[j,k]$, and $PreOVN[j,k]$ for all writers j and k, it chooses the values it will write for the $VN[i,j]$, $PVN[i,j]$, and $OVN[i,j]$, for all writers j as follows:

$VN[i,j]$ Since we want to have $OVN[j,i] = VN[i,j]$ only for writers j whose most recent writes are more recent than the most recent write by writer i, we must choose $VN[i,j] \neq OVN[j,i]$. Similarly, since $PreOVN[j,i]$ is the value that an ongoing write by writer j is planning to write for $OVN[j,i]$, we want to choose $VN[i,j] \neq PreOVN[i,j]$; otherwise we would imply falsely that the ongoing write by writer j had chosen the value it is to write for $OVN[j,i]$ on the basis of the value of $VN[i,j]$ that we are choosing here but have not yet written. Finally, since $VN[i,j]$ is to serve as a "version number" for the current write by writer i, it must be different from the value previously written for $VN[i,j]$. We thus choose the new value for $VN[i,j]$ to be an arbitrary element of the observed set:

$$\{1,2,3,4\} \setminus \{OVN[j,i], PreOVN[j,i], VN[i,j]\}.$$

$PVN[i,j]$ Since we want $PVN[i,j]$ to be the value that was previously written for $VN[i,j]$, we will choose $PVN[i,j]$ to be the observed value for $VN[i,j]$:

$$PVN[i,j] := VN[i,j].$$

$OVN[i,j]$ As was mentioned during the discussion of the version numbers, the values of the $OVN[i,j]$ are to represent the values of the $VN[j,i]$ observed by writer i. Consequently, we assign:

$$OVN[i,j] := VN[j,i].$$

After a writer i performing a write W has chosen the values it is to write for $VN[i,j]$, $PVN[i,j]$, and $OVN[i,j]$, it proceeds to write to its register, in one atomic fell swoop, $Value[i]$, and $VN[i,j]$, $PVN[i,j]$, and $OVN[i,j]$ for all writers j. It is to the "atomic" action of this write to writer i's register that the $Write(W)$ action corresponds.

The $PreOVN[i,j]$ are written somewhat differently. This is so for two reasons. First, since the $PreOVN[i,j]$ are not visible to the readers, it is not necessary to write them with the other values. Second, since it is the purpose of the $PreOVN[i,j]$ to inform other writers of the value of $OVN[i,j]$ that will be written, but has not yet been written, it is vital that the $PreOVN[i,j]$ be written as early as possible. Thus the $PreOVN[i,j]$ are written following the first scan of the writers' registers and following each subsequent scan that returns values different from those returned by the previous scan. Thus each time a scan returns a potentially new set of $VN[j,i]$, we write the new values:

$$PreOVN[i,j] := VN[j,i]$$

for all writers j. The "atomic" action of the last write of the value $PreOVN[i,j]$ as part of the write W corresponds to the action $PWrite(W)_j$.

As was the case with the reader's protocol, a writer performing a write could perform an infinite sequence of scans and never see three consecutive scans return the same values. The solution here is the same as with the reader's protocol. As a writer i performs scans of the writers' registers, it keeps track of those writers that have been seen to change values between scans. As before, if some writer is seen to have changed its values more than once, the last write was performed within the time bounds of writer i's current write. The "atomic" action for writer i's current write may thus be placed immediately before that of the write that was performed within its *Start* and *Finish* bounds; writer i simply terminates its write without changing $Value[i]$, $VN[i,j]$, $PVN[i,j]$, or $OVN[i,j]$. A writer that terminates in this manner is said to have "timed out." Note that since writer i does not change its values while it is scanning (the $PreOVN[i,j]$'s are not compared across scans), and three consecutive, identical scans are needed, the pigeon hole principle dictates a ceiling on the number of scans that a writer need perform that is somewhat different from the corresponding ceiling for readers; after at most $2m+1$ scans, a writer has either seen three consecutive, identical scans or has timed out.

Thus we can summarize the operation of the writer's protocol as follows:

1. A writer performing a write first repeatedly performs scans the of the writers' registers. After each scan (except the first), the values read for the VN, PVN, and OVN are compared to those that were read by the previous scan. If a difference is found, the writer writes out its $PreOVN[i,j]$'s and notes which writers were responsible for the difference. After a sequence of exactly two consecutive identical scans, the $PreOVN$ are read as this may turn out to be the point between the second and third consecutive identical scans.

2. If after $2m + 1$ scans, no three consecutive scans have been observed to have the same values, the write times out by exiting without doing anything further. Otherwise, the third scan of a set of three consecutive, identical scans, along with the last observed set of $PreOVN$, is taken to be a consistent state of the VN, PVN, OVN, and $PreOVN$.

3. New values are now chosen for the $VN[i,j]$, $OVN[i,j]$, and $PVN[i,j]$ according to the rules expressed earlier. After these values have been chosen, they, along with the new value for $Value[i]$ are written to writer i's register in one atomic write.

This completes the discussion of the writer's protocol.

Part II

Code and Counterexamples

7 The Code

Figure 2 presents the code for the reader's protocol published in [PB], rewritten with a bug fix. Similarly, figure 3 presents the code for the published writer's protocol, again rewritten with a bug fix.

A few comments about the code are in order. First note that the actions to which certain key portions of the code correspond have been placed at the right. The $xScan(W)_i$ correspond to the the reads of writer i's register as part of a scan. The only three such scans for which we have explicitly defined actions $xScan(W)_i$ are the last three which are $1Scan(W)_i$, $2Scan(W)_i$, and $3Scan(W)_i$ respectively; since we do not know at the time we perform a scan if it is one of those three scans, we must be content with the variable labels $xScan(W)_i$ in the margin. Similarly for the $xScan(R)_i$. Note that the subscripts that appear in the action labels, such as the i in $xScan(W)_i$, refer to the variables in the code.

Note also that the code for the writer's protocol is specific to writer k; it makes use of the variable k in the code so that it knows the register to which it may write. Note also that the only variables that are shared among the protocols are the VN, PVN, OVN, and $PreOVN$, all other variables are local.

An additional note about the code is that all code within a given pair of $\triangleright\triangleleft$ symbols is to be performed in one atomic action. Thus if a loop is contained within the triangle symbols, the values to be written or read by the loop are written or read all at once; the loop is only notation to quantify what gets written or read.

```
BEGIN                                                                          Start(R)
  Same_Scans := 0; Timed_Out := 0;
  FOR i := 1 TO m DO Changes_Seen[i] := 0; END;
  FOR i := 1 TO m DO
    ▷ FOR j := 1 TO m DO Scan_VN[i,j] := VN[i,j]; END;                         }
      FOR j := 1 TO m DO Scan_OVN[i,j] := OVN[i,j]; END;                       } xScan(R)i
      FOR j := 1 TO m DO Scan_PVN[i,j] := PVN[i,j]; END;                       }
      Scan_Value[i] := Value[i]; ◁                                             }
  END;
  Same_Scans := 1;
  REPEAT
    FOR i := 1 TO m DO
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
    END;
    FOR i := 1 TO m DO
      ▷ FOR j := 1 TO m DO Scan_VN[i,j] := VN[i,j]; END;                       }
        FOR j := 1 TO m DO Scan_OVN[i,j] := OVN[i,j]; END;                     } xScan(R)i
        FOR j := 1 TO m DO Scan_PVN[i,j] := PVN[i,j]; END;                     }
        Scan_Value[i] := Value[i]; ◁                                           }
    END;
    Any_Change_Since_Last_Scan := FALSE;
    FOR i := 1 TO m DO
      i_Changed_Since_Last_Scan := FALSE;
      FOR j := 1 TO m DO
        IF Scan_VN[i,j] ≠ Saved_Scan_VN[i,j]
        THEN i_Changed_Since_Last_Scan := TRUE;
      END;
      IF i_Changed_Since_Last_Scan
      THEN Changes_Seen[i] := Changes_Seen[i] + 1;
        Any_Change_Since_Last_Scan := TRUE;
    END;
    IF Any_Change_Since_Last_Scan
    THEN Same_Scans := 1;
      FOR i := 1 TO m DO
        IF Changes_Seen[i] = 2 THEN Timed_Out := i;
      END;
    ELSE Same_Scans := Same_Scans + 1;
  UNTIL Same_Scans = 2 OR Timed_Out ≠ 0;
  IF Timed_Out ≠ 0
  THEN RETURN(Scan_Value[Timed_Out]);
  ELSE
    FOR i := 1 TO m DO
      N[i] := 1;
      FOR j := 1 TO m DO
        IF Scan_OVN[i,j] ≠ Scan_VN[j,i] AND Scan_OVN[i,j] ≠ Scan_PVN[j,i]
        THEN N[i] := 0;
      END;
      VNS_Size[i] := 0;
      FOR j := 1 TO m DO
        IF Scan_OVN[i,j] = Scan_VN[j,i]
        THEN VNS_Size[i] := VNS_Size[i] + 1;
      END;
    END;
    F := 0; N_plus_VNS_Size := 0;
    FOR i := 1 TO m DO
      IF N[i] + VNS_Size[i] ≥ N_plus_VNS_Size
      THEN F := i; N_plus_VNS_Size := N[i] + VNS_Size[i];
    END;
  RETURN(Scan_Value[F]);
END;                                                                           Finish(R)
```

Figure 2: The reader's protocol.

```
BEGIN                                                                          Start(W)
  Same_Scans := 0; Timed_Out := 0;
  FOR i := 1 TO m DO Changes_Seen[i] := 0; END;
  FOR i := 1 TO m DO
    ▷ FOR j := 1 TO m DO Scan_VN[i,j] := VN[i,j]; END;                         } xScan(W)_i
      FOR j := 1 TO m DO Scan_OVN[i,j] := OVN[i,j]; END;
      FOR j := 1 TO m DO Scan_PVN[i,j] := PVN[i,j]; END;
      Scan_Value[i] := Value[i]; ◁
  END;
  Same_Scans := 1;
  REPEAT
    FOR i := 1 TO m DO
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
    END;
    IF Same_Scans = 1
    THEN
      FOR i := 1 TO m DO
        ▷ PreOVN[k,i] := Scan_VN[i,k]; ◁                                       PWrite(W)_i
      END;
    FOR i := 1 TO m DO
      ▷ FOR j := 1 TO m DO Scan_VN[i,j] := VN[i,j]; END;                       } xScan(W)_i
        FOR j := 1 TO m DO Scan_OVN[i,j] := OVN[i,j]; END;
        FOR j := 1 TO m DO Scan_PVN[i,j] := PVN[i,j]; END;
        Scan_Value[i] := Value[i]; ◁
    END;
    Any_Change_Since_Last_Scan := FALSE;
    FOR i := 1 TO m DO
      i_Changed_Since_Last_Scan := FALSE;
      FOR j := 1 TO m DO
        IF Scan_VN[i,j] ≠ Saved_Scan_VN[i,j]
        THEN i_Changed_Since_Last_Scan := TRUE;
      END;
      IF i_Changed_Since_Last_Scan
      THEN Changes_Seen[i] := Changes_Seen[i] + 1;
        Any_Change_Since_Last_Scan := TRUE;
    END;
    IF Any_Change_Since_Last_Scan
    THEN Same_Scans := 1;
      FOR i := 1 TO m DO
        IF Changes_Seen[i] = 2 THEN Timed_Out := i;
      END;
    ELSE Same_Scans := Same_Scans + 1;
  UNTIL Same_Scans = 3 OR Timed_Out ≠ 0;
  IF Timed_Out ≠ 0
  THEN RETURN;
  ELSE
    FOR i := 1 TO m DO
      ▷ PScan_PreOVN[i,k] := PreOVN[i,k]; ◁                                    PScan(W)_i
    END;
    ▷ FOR i := 1 TO m DO                                                       } Write(W)
        VN[k,i] := Any({1,2,3,4} \ {Scan_VN[k,i], Scan_OVN[i,k], PScan_PreOVN[i,k]});
        OVN[k,i] := Scan_VN[i,k];
        PVN[k,i] := Scan_VN[k,i];
      END;
      Value[k] := VALUE; ◁
  RETURN;
END;                                                                           Finish(W)
```

Figure 3: Writer k's protocol.

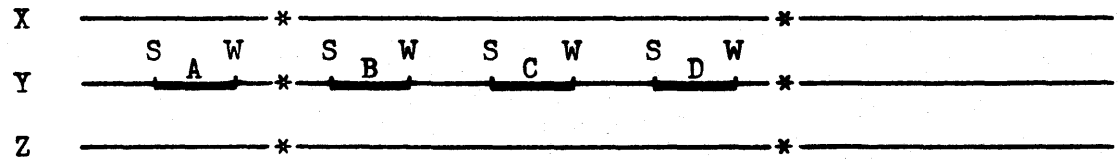

Figure 4:

8 The First Counterexample

Let us first assume that the writer's protocol maintians a consistent state of the world; that atomic write points may be inserted within the bounds of each write such that the value of F is a constant between those points, and at each point p, the value of F at p is the writer that performed the write whose atomic point most recently preceeds p.

Thus if a read R is performed in an interval containing no atomic write points, we can place an atomic read point anywhere between $Start(R)$ and $Finish(R)$, and R will necessarily return the value written by the write whose atomic write point most recently preceeds R's atomic read point. Similarly, for reads R that time out, we have argued that R must return the value of a write that was performed completely within the bounds of $Start(R)$ and $Finish(R)$; if the atomic read point for R is placed immediately after that of the atomic write point of the contained write, then again R necessarily returns the value written by the write whose atomic write point most recently preceeds its own atomic read point.

Unfortunately, it is not the case that all reads either are performed in write-free intervals or explicitly time out, as figure 4 illustrates. Figure 4 shows the actions of three writers labeled X, Y, and Z; we will assume in these figures that the writers are presented in increasing order, thus $X < Y < Z$. In the interval pictured, X and Z do not write while Y writes four times. The *Scan* and $Write$ actions of the writes are indicated by the points labeled by S and W respectively. Note that under S we are lumping together all three consecutive, identical scans made by a writer, as well as the $PWrite$ action. Also included in the diagram are two scans of the three writers' registers made by a reader as part of a single read R. The * signs denote the atomic read points of the reads of the individual writers' registers performed as part of the scans. Thus writer Y starts with a complete write A. This is followed by the complete first scan of the read R. This is then followed by three more complete writes by writer Y and the final scan of R.

Write A sees the current VN's posted by all three writers and records them as its

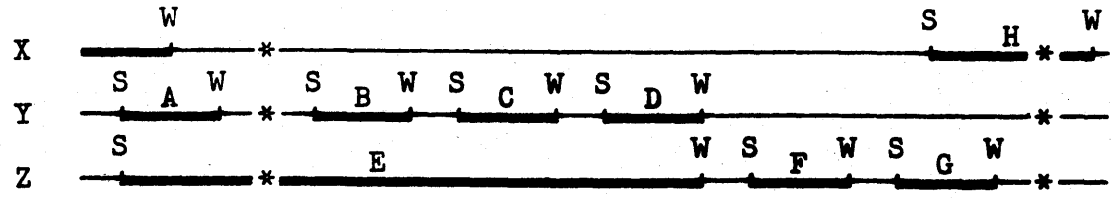

Figure 5:

$OVN[i,j]$'s when it writes, while changing its own set of $VN[i,j]$'s. At this point, the state of the world is seen by the first scan of read R. Write B then writes a new set of $VN[i,j]$'s which by choice must differ from those written by write A. If the second scan of R is to read the same VN's as the first scan we see that writer Y must write again (indeed twice since the protocol requires a minimum of three writes for a writer to restore its VN for itself) to restore the VN's that had been written as part of write A. This having been accomplished, the second scan of read R is performed and returns the same state of the world as was seen by the first scan of R. Thus the reader performing read R cannot tell that a write has occured between the two read scans, although several have, and proceeds to return a value based upon the information observed by the two scans.

One may ask if the value returned in the above example will violate the atomicity requirements for the three-writer register construction. In this case, the answer is that the value returned is legitimate. The value returned is that written by write D. Since write D is completely contained within the bounds of read R, its atomic action is as well, and as in the case of the timed out reads, it is legitimate to place the atomic read action of R immediately following the atomic write action of D. In [PB], R is referred to as having timed out without knowing that it did so. That paper then attempts to generalize the argument, used above to demonstrate the need for C and D if the scans of R are to agree, to provide a proof that when a writer times out without knowing it has done so, it still returns a correct value. It was the study of that proof that led to the development of the first counterexample to the correctness of the algorithm, thus it is instructive to repeat it here.

Given the last two scans of a read R as shown in figure 5, assume that the values of the VN's seen by the two scans are identical. Now divide the writers into two sets, the "changing" writers that performed the *Write* action of some write between the two scans of R, and the "unchanging" writers that did not perform the *Write* action of any write between the two scans of R. By that definition, writers Y and Z are changing writers while writer X is an unchanging writer in figure 5. Now by reasoning presented

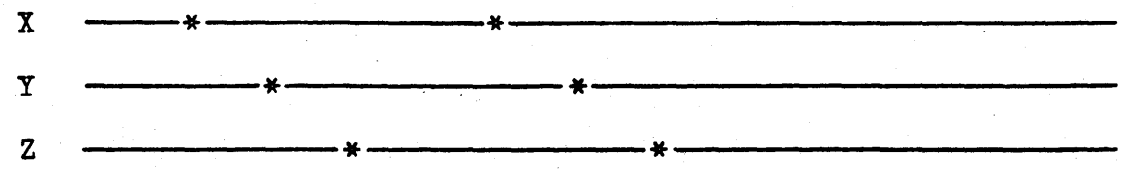

Figure 6:

earlier, if the two scans of R are to see the same VN's for all writers, writes C and D must occur between $Write(B)$ and the second scan of read R; in general, every changing writer must perform a complete write between the two scans of R. Thus at the second scan of R, all of the changing writers will be observed to have "seen" the VN's of the unchanging writers whereas the unchanging writers will be observed not to "see" the VN's of any of the changing writers. Also, since each changing writer has written at least twice between the most recent write by any unchanging writer and the second scan of R, we should have $N(i) = 0$ for all unchanging writers i. Thus it is completely impossible for the value of an unchanging writer to be returned if there exist any changing writers. If the value returned by R is read from the register of a changing writer, then it was written by a write that occured entirely between the two scans of R. If the value returned is read from the register of an unchanging writer, then there are no changing writers, and the last two scans of R occured in an interval in which no writing took place. Thus R returns a legitimate value.

The problem with this proof is shown in figure 6 which demonstrates the real picture of how read scans occur. The notions of "the point at which the first scan of R occured" and thus of "changing" and "unchanging" writers, are therefore not well defined. Suppose the following definition of "changing" writer is made to eliminate ambiguity: a writer i will be defined to be a changing writer if it completed a write W between the reads of its register in the first and second consecutive, identical scans made by the read R; that is, if $1Scan(R)_i < Write(W) < 2Scan(R)_i$. Thus in figure 7, writer Z is a changing writer while writers X and Y are not. The same reasoning as above then shows that some writes C and D must occur between $Write(B)$ and the read, $2Scan(R)_Z$, of writer Z's register in the second scan.

There is a problem with this however, that is demonstrated by figure 8. Assume that the scans of the read R see the same VN's. Writer X is a changing writer while writer Y is an unchanging writer. Writer Y will be seen to have observed the VN's written by writer X during the write D. Writer X, on the other hand, will be observed to have seen the VN's written by writer Y prior to the write E. Writer Y will consequently be

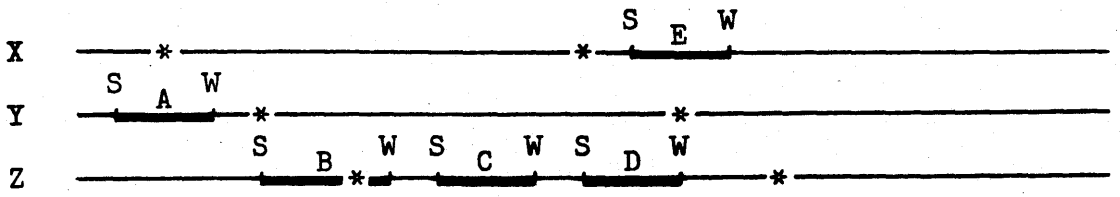

Figure 7:

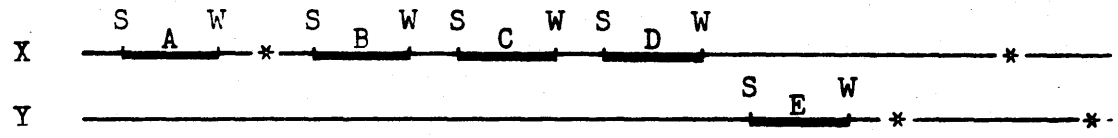

Figure 8:

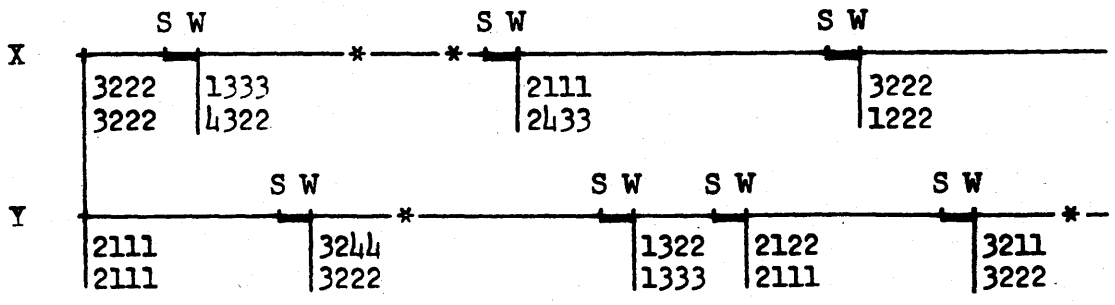

Figure 9: The first counterexample.

judged, correctly, to be the writer that wrote more recently before the second scan of R, and its value, that written by E, will be returned by R. Read R thus returns the value written by an *unchanging* writer despite the existence of a changing writer. Clearly, the reasoning sketched above no longer works; one then asks if a conterexample may be constructed to the algorithm in a similar manner.

The answer to this question is that we can. Such a counterexample is listed in figure 9. The numbers following the vertical lines are the values of the various variables following the actions to which the vertical lines are connected; the numbers below the horizontal time-line for writer X refer, in order, to the $VN[X,i]$, $PVN[X,i]$, $OVN[X,i]$, and $PreOVN[X,i]$; the rows of numbers are presented in the same order as the time-lines for the different writers. For example, following the first write by writer X, we have,

$$\begin{aligned} VN[X,X] = 1 \quad &and \quad VN[X,Y] = 4 \\ PVN[X,X] = 3 \quad &and \quad PVN[X,Y] = 3 \\ OVN[X,X] = 3 \quad &and \quad OVN[X,Y] = 2 \\ PreOVN[X,X] = 3 \quad &and \quad PreOVN[X,Y] = 2. \end{aligned}$$

Then what this counterexample has done is to perform, without interruption, the first scan of the read R as well as the read of writer X's register for the second scan of R. Before the second scan of R gets to read the value in Y's register, however, we have performed a series of writes that render completely meaningless the first values read. In particular, we have written so that the values of $VN[Y,X]$ and $VN[Y,Y]$ observed by the second scan equal the values of these variables observed by the first scan; this implies that the read R detects no writes occuring between its scans and will select a value to return based on the values seen by the second scan. But for the values returned by the second scan we have:

$$1 = OVN[Y,X] \neq VN[X,Y] = 4 \quad and \quad 1 = OVN[Y,X] \neq PVN[X,Y] = 3$$

and

$$2 = OVN[Y,Y] \neq VN[Y,Y] = 3 \quad \textit{and} \quad 1 = OVN[Y,X] \neq VN[X,Y] = 4$$

implying that $N(Y) = 0$ and $|VNS(Y)| = 0$. Also,

$$3 = OVN[X,X] = PVN[X,X] = 3 \quad \textit{and} \quad 2 = OVN[X,Y] = PVN[Y,X] = 2$$

implying that $N(X) = 1$ while $|VNS(X)| = 0$. The value of F computed on the basis of these values is $F = X$. Thus the read R will return the value read from the register of writer X during its second scan. Since this value was written by the first write shown for writer X, and the atomic write action of the first write shown for writer Y must be interposed between the atomic write action of the first write shown for writer X and the first scan of R, the atomicity condition is violated.

One will note that the first and second scans did not observe the same values for $OVN[Y,X]$. One might ask then if the algorithm would perform correctly if not only the VN's, but the PVN's and OVN's as well were required to be constant across the two scans of a read. A counterexample communicated by Burns shows that both scans of a read R may see the same values for the VN's, PVN's, and OVN's, and still return a value that is no longer valid.

9 The Second Counterexample

In our discussion of the previous counterexample, we assumed that the writers write in a manner that respects the atomicity condition. This turns out not to be so, the result being another counterexample to the correctness of the algorithm.

Recall that when a writer is reading the values that it needs to determine what to write, it reads the OVN's before the $PreOVN$'s. At the same time, however, writers write their $PreOVN$'s before they write their OVN's. This leads to trouble.

Figure 10 presents an example of how this fact can result in the improper execution of the algorithm. The second write by writer X scans the value $OVN[Y,X]$ before the write point of the first write by writer Y. Before the second write by writer X gets around to reading $PreOVN[Y,X]$ (at the point marked "PS"), however, writer Y both writes and scans; the write by writer Y invalidates the value of $OVN[Y,X]$ seen by writer X while the scan invalidates the value of $PreOVN[Y,X]$. This means that the second write by writer X completely fails to see the value of $OVN[Y,X]$ written by the first write by writer Y.

Let P be the point immediately preceeding the *Write* action of the second write by writer X. Let Q be the point immediately following the same action.

We have the following set of equations at P:

$$3 = OVN[X,X] = PVN[X,X] = 3 \neq VN[X,X] = 4$$

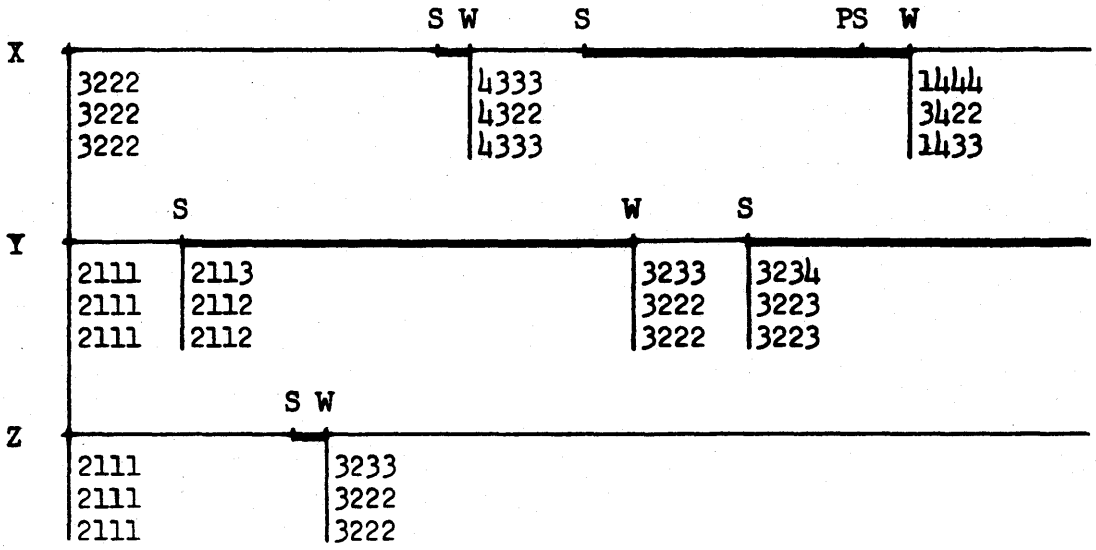

Figure 10: The second counterexample.

$$2 = OVN[X,Y] = PVN[Y,X] = 2 \neq VN[Y,X] = 3$$

$$3 = OVN[X,Z] = VN[Z,X] = 3$$

Thus $N(X) = 1$ and $|VNS(X)| = 1$.

$$3 = OVN[Y,X] = PVN[X,Y] = 3 \neq VN[X,Y] = 4$$

$$2 = OVN[Y,Y] = PVN[Y,Y] = 2 \neq VN[Y,Y] = 3$$

$$2 = OVN[Y,Z] = PVN[Z,Y] = 2 \neq VN[Z,Y] = 3$$

Thus $N(X) = 1$ and $|VNS(X)| = 0$.

$$3 = OVN[Z,X] = PVN[X,Z] = 3 \neq VN[X,Z] = 4$$

$$2 = OVN[Z,Y] = PVN[Y,Z] = 2 \neq VN[Y,Z] = 3$$

$$2 = OVN[Z,Z] = PVN[Z,Z] = 2 \neq VN[Z,Z] = 3$$

Thus $N(X) = 1$ and $|VNS(X)| = 0$. Consequently, $F = X$ at P.

We have the following set of equations at Q:

$$4 = OVN[X,X] = PVN[X,X] = 4 \neq VN[X,X] = 1$$

$$2 = OVN[X,Y] = PVN[Y,X] = 2 \neq VN[Y,X] = 3$$

$$3 = OVN[X,Z] = VN[Z,X] = 3$$

Thus $N(X) = 1$ and $|VNS(X)| = 1$.

$$3 = OVN[Y,X] = VN[X,Y] = 3$$

$$2 = OVN[Y,Y] = PVN[Y,Y] = 2 \neq VN[Y,Y] = 3$$

$$2 = OVN[Y,Z] = PVN[Z,Y] = 2 \neq VN[Z,Y] = 3$$

Thus $N(X) = 1$ and $|VNS(X)| = 1$.

$$3 = OVN[Z,X] \neq PVN[X,Z] = 4 and 3 = OVN[Z,X] \neq VN[X,Z] = 1$$

$$2 = OVN[Z,Y] = PVN[Y,Z] = 2 \neq VN[Y,Z] = 3$$

$$2 = OVN[Z,Z] = PVN[Z,Z] = 2 \neq VN[Z,Z] = 3$$

Thus $N(X) = 0$ and $|VNS(X)| = 0$. Consequently, since $Y > X$, $F = Y$ at P.

This is not good because it implies that the most recent atomic write action preceeding P is not that of the first write by writer Y whereas the most recent atomic write action preceeding Q is that of the first write by writer Y. Thus these writes were not performed in a simulated atomic manner.

The obvious fix to this problem is to scan the $PreOVN$ values earlier. The code for the writer's protocol that is proved correct in the next part of this paper performs the scan of the $PreOVN$ values between the second and third consecutive identical scans of the writers' registers instead of after all three consecutive identical scans have completed.

Part III

Code and Correctness

10 The Code

The code for the algorithm we will be proving correct is found in figures 11 and 12. Note that the only differences between this code and that which was presented earlier are: the number of consecutive, identical scans a reader makes is now three; all of the VN's, PVN's, and OVN's are now compared between scans for both reads and writes; and writes perform their final reads of the $PreOVN$'s between their second and third consecutive, identical scans. The first two of these were suggested by Burns as corrections to eliminate the first counterexample. The third is a fix to eliminate the conditions that led to the second counterexample.

```
BEGIN                                                                    Start(R)
  Same_Scans := 0; Timed_Out := 0;
  FOR i := 1 TO m DO Changes_Seen[i] := 0; END;
  FOR i := 1 TO m DO
    ▷ FOR j := 1 TO m DO Scan_VN[i,j] := VN[i,j]; END;                   }
      FOR j := 1 TO m DO Scan_OVN[i,j] := OVN[i,j]; END;                 } xScan(R)i
      FOR j := 1 TO m DO Scan_PVN[i,j] := PVN[i,j]; END;                 }
      Scan_Value[i] := Value[i]; ◁                                       }
  END;
  Same_Scans := 1;
  REPEAT
    FOR i := 1 TO m DO
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
    END;
    FOR i := 1 TO m DO
      ▷ FOR j := 1 TO m DO Scan_VN[i,j] := VN[i,j]; END;                 }
        FOR j := 1 TO m DO Scan_OVN[i,j] := OVN[i,j]; END;               } xScan(R)i
        FOR j := 1 TO m DO Scan_PVN[i,j] := PVN[i,j]; END;               }
        Scan_Value[i] := Value[i]; ◁                                     }
    END;
    Any_Change_Since_Last_Scan := FALSE;
    FOR i := 1 TO m DO
      i_Changed_Since_Last_Scan := FALSE;
      FOR j := 1 TO m DO
        IF Scan_VN[i,j] ≠ Saved_Scan_VN[i,j] OR
           Scan_OVN[i,j] ≠ Saved_Scan_OVN[i,j] OR
           Scan_PVN[i,j] ≠ Saved_Scan_PVN[i,j]
        THEN i_Changed_Since_Last_Scan := TRUE;
      END;
      IF i_Changed_Since_Last_Scan
      THEN Changes_Seen[i] := Changes_Seen[i] + 1;
           Any_Change_Since_Last_Scan := TRUE;
    END;
    IF Any_Change_Since_Last_Scan
    THEN Same_Scans := 1;
      FOR i := 1 TO m DO
        IF Changes_Seen[i] = 2 THEN Timed_Out := i;
      END;
    ELSE Same_Scans := Same_Scans + 1;
  UNTIL Same_Scans = 3 OR Timed_Out ≠ 0;
  IF Timed_Out ≠ 0
  THEN RETURN(Scan_Value[Timed_Out]);
  ELSE
    FOR i := 1 TO m DO
      N[i] := 1;
      FOR j := 1 TO m DO
        IF Scan_OVN[i,j] ≠ Scan_VN[j,i] AND Scan_OVN[i,j] ≠ Scan_PVN[j,i]
        THEN N[i] := 0;
      END;
      VNS_Size[i] := 0;
      FOR j := 1 TO m DO
        IF Scan_OVN[i,j] = Scan_VN[j,i]
        THEN VNS_Size[i] := VNS_Size[i] + 1;
      END;
    END;
    F := 0; N_plus_VNS_Size := 0;
    FOR i := 1 TO m DO
      IF N[i] + VNS_Size[i] ≥ N_plus_VNS_Size
      THEN F := i; N_plus_VNS_Size := N[i] + VNS_Size[i];
    END;
  RETURN(Scan_Value[F]);
END                                                                      Finish(R)
```

Figure 11: The reader's protocol.

```
BEGIN                                                                          Start(W)
  Same_Scans := 0; Timed_Out := 0;
  FOR i := 1 TO m DO Changes_Seen[i] := 0; END;
  FOR i := 1 TO m DO
    ▷ FOR j := 1 TO m DO Scan_VN[i,j] := VN[i,j]; END;                         }
      FOR j := 1 TO m DO Scan_OVN[i,j] := OVN[i,j]; END;                       } xScan(W)_i
      FOR j := 1 TO m DO Scan_PVN[i,j] := PVN[i,j]; END;                       }
      Scan_Value[i] := Value[i]; ◁                                             }
  END;
  Same_Scans := 1;
  REPEAT
    FOR i := 1 TO m DO
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
      FOR j := 1 TO m DO Saved_Scan_VN[i,j] := Scan_VN[i,j]; END;
    END;
    IF Same_Scans = 1
    THEN
      FOR i := 1 TO m DO
        ▷ PreOVN[k,i] := Scan_VN[i,k]; ◁                                       PWrite(W)_i
      END;
    ELSIF Same_Scans = 2
    THEN
      FOR i := 1 TO m DO
        ▷ PScan_PreOVN[i,k] := PreOVN[i,k]; ◁                                  PScan(W)_i
      END;
    FOR i := 1 TO m DO
      ▷ FOR j := 1 TO m DO Scan_VN[i,j] := VN[i,j]; END;                       }
        FOR j := 1 TO m DO Scan_OVN[i,j] := OVN[i,j]; END;                     } xScan(W)_i
        FOR j := 1 TO m DO Scan_PVN[i,j] := PVN[i,j]; END;                     }
        Scan_Value[i] := Value[i]; ◁                                           }
    END;
    Any_Change_Since_Last_Scan := FALSE;
    FOR i := 1 TO m DO
      i_Changed_Since_Last_Scan := FALSE;
      FOR j := 1 TO m DO
        IF Scan_VN[i,j] ≠ Saved_Scan_VN[i,j] OR
           Scan_OVN[i,j] ≠ Saved_Scan_OVN[i,j] OR
           Scan_PVN[i,j] ≠ Saved_Scan_PVN[i,j]
        THEN i_Changed_Since_Last_Scan := TRUE;
      END;
      IF i_Changed_Since_Last_Scan
      THEN Changes_Seen[i] := Changes_Seen[i] + 1;
        Any_Change_Since_Last_Scan := TRUE;
    END;
    IF Any_Change_Since_Last_Scan
    THEN Same_Scans := 1;
      FOR i := 1 TO m DO
        IF Changes_Seen[i] = 2 THEN Timed_Out := i;
      END;
    ELSE Same_Scans := Same_Scans + 1;
  UNTIL Same_Scans = 3 OR Timed_Out ≠ 0;
  IF Timed_Out ≠ 0
  THEN RETURN;
  ELSE
    ▷ FOR i := 1 TO m DO                                                       }
        VN[k,i] := Any({1,2,3,4} \ {Scan_VN[k,i], Scan_OVN[i,k], PScan_PreOVN[i,k]}); }
        OVN[k,i] := Scan_VN[i,k];                                              } Write(W)
        PVN[k,i] := Scan_VN[k,i];                                              }
      END;                                                                     }
      Value[k] := VALUE; ◁                                                     }
  RETURN;
END;                                                                           Finish(W)
```

Figure 12: Writer k's protocol.

Note that up to this point we have ignored the question of initial values. We will start the composition automaton in a state in which no readers or writers are reading or writing and for which:

$$VN[i,j] = 2$$

$$PVN[i,j] = 1$$

$$PreOVN[i,j] = OVN[i,j] = 1$$

for all $i > 1$, and for which

$$VN[1,j] = 3$$

$$PVN[1,j] = 2$$

$$PreOVN[1,j] = OVN[1,j] = 2.$$

We will also assume that this configuration was reached by performing a number of writes, at least one per writer, building on a previous set of values. As such the most recent write W by writer i for which $Write(W) < s$ is well defined for all states s and all writers i.

11 Definitions

Let us begin our proof of correctness by recapitulating the definitions of the preceeding sections.

DEFINITION: Let W be any write of a value to the composition automaton and R be any read of the value in the composition automaton. Then $Value(W)$ and $Value(R)$ refer to the values written by W and read by R respectively.

DEFINITION: Let W be any write by writer i. Then the following actions are associated with W:

$Start(W)$ The request to writer i to begin the write W. This is the first action in the write W.

$Finish(W)$ Acknowledgement that the write W has just completed. This is the last action in the write W.

DEFINITION: Let W be any write by writer i such that W did not time out. Then in addition to the above actions, the following actions are associated with W:

$1Scan(W)_j$ The atomic action associated with the read of writer j's register during the first of the last three scans performed by writer i as part of W. Note that we are actually defining the m separate actions:

$$1Scan(W)_1 < 1Scan(W)_2 < \ldots < 1Scan(W)_m.$$

$PWrite(W)_j$ The atomic action associated with the last write of $PreOVN[i,j]$ by writer i as part of W. Note again that we are defining m separate actions. It is not necessary, however, to perform the writes of the $PreOVN[i,j]$ separately; since the values are all being written to the same register, it would be quite legitimate to write them all at once. The algorithm just happens to write them separately.

$2Scan(W)_j$ The atomic action associated with the read of writer j's register during the second of the last three scans performed by writer i as part of W. Note again that we are defining m separate actions.

$Scan(W)$ An action inserted immediately following $2Scan(W)_m$. The significance of this action will be defined later.

$PScan(W)_j$ The atomic action associated with the last read of $PreOVN[j,i]$ from writer j's register performed by writer i as part of W. Note again that we are defining m separate actions.

$3Scan(W)_j$ The atomic action associated with the read of writer j's register during the last scan performed by writer i as part of W. Note again that we are defining m separate actions.

$Write(W)$ The atomic action associated with the write of $Value(W)$ and new VN's, OVN's, and PVN's to writer i's register as part of the write W.

Note then that for a write W by writer i that does not time out, the actions of W defined above occur in the following order:

$$\begin{aligned} Start(W) < {} & 1Scan(W)_1 < \ldots < 1Scan(W)_m < \\ & PWrite(W)_1 < \ldots < PWrite(W)_m < \\ & 2Scan(W)_1 < \ldots < 2Scan(W)_m < \\ & Scan(W) < \\ & PScan(W)_1 < \ldots < PScan(W)_m < \\ & 3Scan(W)_1 < \ldots < 3Scan(W)_m < \\ & Write(W) < Finish(W) \end{aligned}$$

DEFINITION: Let R be any read by reader i. Then the following actions are associated with R:

$Start(R)$ The request to reader i to begin the read R. This is the first action in the read R.

$Finish(R)$ Acknowledgement that the read R has just completed. This is the last action in the read R.

DEFINITION: Let R be any read by reader i such that R did not time out. Then in addition to the above actions, the following actions are associated with R:

$1Scan(R)_j$ The atomic action associated with the read of writer j's register during the first of the last three scans performed by reader i as part of R. Note that we are actually defining the m separate actions:

$$1Scan(R)_1 < 1Scan(R)_2 < \ldots < 1Scan(R)_m.$$

$2Scan(R)_j$ The atomic action associated with the read of writer j's register during the second of the last three scans performed by reader i as part of R. Note again that we are defining m separate actions.

$3Scan(R)_j$ The atomic action associated with the read of writer j's register during the last scan performed by reader i as part of R. Note again that we are defining m separate actions.

Note that for a read R by reader i that does not time out, the actions defined above occur in the following order:

$$\begin{aligned} Start(R) \; < \; & 1Scan(R)_1 < \ldots < 1Scan(R)_m < \\ & 2Scan(R)_1 < \ldots < 2Scan(R)_m < \\ & 3Scan(R)_1 < \ldots < 3Scan(R)_m < Finish(R) \end{aligned}$$

DEFINITION: Let s be any state in an execution of the composition automaton. Let j and k be any writers.

$VN[j,k]_s$ Is the value of $VN[j,k]$ at state s. It is important to note that this value is stored in writer j's register and that it remains constant between consecutive $Write(W_j)$ actions of writes W_j by writer j.

$OVN[j,k]_s$ Is the value of $OVN[j,k]$ at state s. Again, it is important to note that this value is stored in writer j's register and that it remains constant between consecutive $Write(W_j)$ actions of writes W_j by writer j.

$PVN[j,k]_s$ Is the value of $PVN[j,k]$ at state s. Again, it is important to note that this value is stored in writer j's register and that it remains constant between consecutive $Write(W_j)$ actions of writes W_j by writer j.

$Value[j]_s$ If W_j is the last write by writer j for which $Write(W_j) < s$ then we define $Value[j]_s$ to be $Value(W_j)$. Again, this value is stored in writer j's register and changes only at the points $Write(W)$ for writes W by writer j.

$PreOVN[j,k]_s$ Is the value of $PreOVN[j,k]$ at state s. It is important to note that this value is stored in writer j's register, that it is visible only to the writers and not to the readers, and that it changes between some scans of a write W_j by writer j. It remains constant, however, for all states between $PWrite(W_j)_k$ and $Finish(W_j)$ for all writers k and all writes W_j by writer j.

DEFINITION: Let W be a write by writer i that does not time out. Let j and k be writers. Define $VN[j,k]_W$, $OVN[j,k]_W$, and $PVN[j,k]_W$ to be the values of $VN[j,k]$, $OVN[j,k]$, and $PVN[j,k]$ respectively, observed by the last three scans of W. Thus if s, t, and u are the states following $1Scan(W)_j$, $2Scan(W)_j$, and $3Scan(W)_j$ respectively, then we have:

$$VN[j,k]_W = VN[j,k]_s = VN[j,k]_t = VN[j,k]_u$$
$$OVN[j,k]_W = OVN[j,k]_s = OVN[j,k]_t = OVN[j,k]_u$$
$$PVN[j,k]_W = PVN[j,k]_s = PVN[j,k]_t = PVN[j,k]_u$$

Define $PreOVN[j,k]_W$ to be the value of $PreOVN[j,k]$ observed by the write W. Thus if v is the state following $PScan(W)_j$ then we have $PreOVN[j,k]_W = PreOVN[j,k]_v$.

DEFINITION: Let R be a read by reader i that does not time out. Let j and k be writers. Define $VN[j,k]_R$, $OVN[j,k]_R$, and $PVN[j,k]_R$ to be the values of $VN[j,k]$, $OVN[j,k]$, and $PVN[j,k]$ respectively, observed by the last three scans of R. Thus if s, t, and u are the states following $1Scan(R)_j$, $2Scan(R)_j$, and $3Scan(R)_j$ respectively, then we have:

$$VN[j,k]_R = VN[j,k]_s = VN[j,k]_t = VN[j,k]_u$$
$$OVN[j,k]_R = OVN[j,k]_s = OVN[j,k]_t = OVN[j,k]_u$$
$$PVN[j,k]_R = PVN[j,k]_s = PVN[j,k]_t = PVN[j,k]_u$$

The following lemma embodies the rules by which the $VN[i,j]$, $OVN[i,j]$, $PVN[i,j]$, and $PreOVN[i,j]$ are picked each time a writer writes.

Lemma 1 *Let W be a write that does not time out and let i be the writer that performed the write W. Let j be any writer. Let s, t, u, and v be the states following $PScan(W)_j$, $3Scan(W)_j$, $3Scan(W)_i$, and $Write(W)$ respectively. Then the following hold:*

$$\begin{aligned} VN[i,j]_v &\neq VN[i,j]_u \\ VN[i,j]_v &\neq OVN[j,i]_t \\ VN[i,j]_v &\neq PreOVN[j,i]_s \\ OVN[i,j]_v &= VN[j,i]_t \\ PVN[i,j]_v &= VN[i,j]_u. \end{aligned}$$

Also, let x be the state following $PWrite(W)_j$. Then

$$PreOVN[i,j]_x = VN[j,i]_W = VN[j,i]_t = OVN[i,j]_v.$$

Proof of Lemma 1: This follows directly from the definitions of the $PScan$, $3Scan$, and $Write$ actions and from trivial examination of the code. □

Note that $VN[i,j]_v \neq VN[i,j]_u$ implies that a writer changes all of its VN's every time that it performs a write that does not time out.

DEFINITION: Let i be a writer and let s be a state in an execution of the composition automaton. Then we will define:

$$VNS(i)_s = \{j | 1 \leq j \leq m, OVN[i,j]_s = VN[j,i]_s\}.$$

Let i be a writer and let R be any read that did not time out. We will define:

$$VNS(i)_R = \{j | 1 \leq j \leq m, OVN[i,j]_R = VN[j,i]_R\}.$$

DEFINITION: Let i be a writer and let s be a state in an execution of the composition automaton. Then we will define:

$$N(i)_s = \begin{cases} 1 & \text{if for all writers } j, \text{ either} \\ & OVN[i,j]_s = VN[j,i]_s \text{ or } OVN[i,j]_s = PVN[j,i]_s \\ 0 & \text{otherwise.} \end{cases}$$

Let i be a writer and let R be any read that did not time out. We will define:

$$N(i)_R = \begin{cases} 1 & \text{if for all writers } j, \text{ either} \\ & OVN[i,j]_R = VN[j,i]_R \text{ or } OVN[i,j]_R = PVN[j,i]_R \\ 0 & \text{otherwise.} \end{cases}$$

DEFINITION: Let s be a state in an execution of the composition automaton. Then we will define:

$$F(s) = MAX\{i | 1 \leq i \leq m, |VNS(i)_s| + N(i)_s = MAX_{1 \leq j \leq m}(|VNS(j)_s| + N(j)_s)\}.$$

Let R be any read that did not time out. We will define:

$$F(R) = MAX\{i | 1 \leq i \leq m, |VNS(i)_R| + N(i)_R = MAX_{1 \leq j \leq m}(|VNS(j)_R| + N(j)_R)\}.$$

Recall that the value of $F(s)$ may be thought of as the writer whose 1-writer $n+m$-reader register contains the current value for the m-writer n-reader register.

12 Basic Facts

Most of the following theorems, lemmas, corollaries, and such are useful in understanding how writers, writing according to the writer's protocol, are able to write in such a way that $F(s)$ may always be taken to be the "current" value of the m-writer n-reader atomic register.

The following lemma establishes a little fact that will be used throughout the remainder of this paper.

Lemma 2 *For all writers i and all states s in an execution of the composition automaton, $i \notin VNS(i)_s$.*

Proof of Lemma 2: Let i be any writer and s be any state in an execution of the composition automaton. Let W_i be the last write by writer i such that $Write(W_i) < s$. Let t and u be the states following $3Scan(W_i)_i$ and $Write(W_i)$ respectively. Then by Lemma 1 we have $VN[i,i]_u \neq VN[i,i]_t = OVN[i,i]_u$. By choice of W_i, the values of $VN[i,i]$ and $OVN[i,i]$ in writer i's register remain constant between u and s and thus $VN[i,i]_s = VN[i,i]_u$ and $OVN[i,i]_s = OVN[i,i]_u$. Thus $VN[i,i]_s \neq OVN[i,i]_s$ and by definition of $VNS(i)_s$ we have $i \notin VNS(i)_s$ as desired. □

All of the actions we have just described refer to particular, meaningful operations performed during an execution of the read or write protocols, with one exception. In particular, $Scan(W)$ for a write W that did not time out was defined to be an action inserted immediately after $2Scan(W)_m$ but it has had no meaning assigned to it. We will give it meaning by showing that the values of the VN's, OVN's, and PVN's observed by the last three scans of W are identical to those in the writers' registers in the state following $Scan(W)$; if u is the state following $Scan(W)$ then $VN[j,k]_u = VN[j,k]_W$, $OVN[j,k]_u = OVN[j,k]_W$, and $PVN[j,k]_u = PVN[j,k]_W$ for all writers j and k. Thus the values seen by the last three scans made during the write W may be thought to have been read by an atomic scan at the point $Scan(W)$. This is demonstrated by the following Lemmas and Corollary.

Lemma 3 *Let i and j be any writers. Let s and t be any two states, $s < t$, in an execution of the composition automaton. If $VN[i,j]_s = VN[i,j]_t$ and there exists some write W by writer i such that $s < Write(W) < t$ then there exists at least one write W_1 by writer i such that*

$$s < Scan(W_1) < Write(W_1) < t.$$

If $i = j$ then there exist at least two writes W_1 and W_2 by writer i such that

$$s < Scan(W_1) < Write(W_1) < Scan(W_2) < Write(W_2) < t.$$

Proof of Lemma 3: Let W_0 be the first write by writer i such that $s < Write(W_0) < t$. Let u be the state following $Write(W_0)$. Then by the way the VN's and PVN's are chosen (ie. Lemma 1), we have

$$VN[i,j]_u \neq PVN[i,j]_u = VN[i,j]_s.$$

Now since $VN[i,j]_t = VN[i,j]_s$ there must be another write by writer i between u and t to bring the value of $VN[i,j]$ back to what it was at s. Let W_1 be the first such write. Since W_1 must start after W_0 finished, we have $s < u < Scan(W_1) < Write(W_1) < t$ and W_1 is as desired.

In the event that $i = j$, we have additionally, by Lemma 1, that $OVN[i,i]_u = VN[i,i]_s$. Thus if v is the state following $Write(W_1)$, by the way VN's are chosen we have:

$$VN[i,i]_v \neq OVN[i,i]_u = VN[i,i]_s.$$

Again, since $VN[i,i]_t = VN[i,i]_s$, there must be yet another write by writer i between v and t to bring the value of $VN[i,i]$ back to what it was at s. Let W_2 be the first such write. Again, since W_2 must start after W_1 finished, we have $s < Scan(W_1) < Write(W_1) < v < Scan(W_2) < Write(W_2) < t$, and W_1 and W_2 are as desired. □

Lemma 4 *Let W be any write by a writer i such that W did not time out. Then there does not exist a writer j and a write W_j by writer j such that $2Scan(W)_j < Write(W_j) < 3Scan(W)_j$.*

Proof of Lemma 4: Assume otherwise and let j be a writer for which there exists a write W_j such that $2Scan(W)_j < Write(W_j) < 3Scan(W)_j$. Let s and t be the states following $2Scan(W)_j$ and $3Scan(W)_j$ respectively. Then since the last three scans of W saw the same values in the registers, we have $VN[j,k]_W = VN[j,k]_s = VN[j,k]_t$ for all writers k implying that $VN[j,i]_s = VN[j,i]_t$. Now we have assumed that there is a write W_j by writer j for which $s < Write(W_j) < t$, so by Lemma 3, there exists some write W_j' by writer j such that $s < Scan(W_j') < Write(W_j') < t$; let W_j' be the last such write. If v is the state following $Write(W_j')$, then by choice of W_j', $VN[j,i]$ remains constant between v and t implying $VN[j,i]_v = VN[j,i]_t$. Let x be the state following $PScan(W_j')_i$ and note that

$$PWrite(W)_j < 2Scan(W)_j < Scan(W_j') < x < Write(W_j') < 3Scan(W)_j.$$

Then since $PreOVN[i,j]$ remains constant between $PWrite(W)_j$ and $3Scan(W)_j$, by Lemma 1 we have $PreOVN[i,j]_x = VN[j,i]_W = VN[j,i]_t$. Also, by Lemma 1 we have $VN[j,i]_v \neq PreOVN[i,j]_x$. But this implies $VN[j,i]_v \neq PreOVN[i,j]_x = VN[j,i]_t$ contradicting the $VN[j,i]_v = VN[j,i]_t$ we saw above. Thus our assumption is incorrect and the Lemma is proved. □

Corollary 5 *Let W be any write by writer j such that W did not time out. Let u be the state following $Scan(W)$. Then $VN[j,k]_u = VN[j,k]_W$, $OVN[j,k]_u = OVN[j,k]_W$, and $PVN[j,k]_u = PVN[j,k]_W$ for all writers j and k.*

Proof of Corollary 5: By Lemma 4, there are no writes to writer j's register that could change the values of $VN[j,k]$, $OVN[j,k]$, and $PVN[j,k]$ between $2Scan(W)_j$ and $3Scan(W)_j$ for any writer k. Thus if s and t are the states following $2Scan(W)_j$ and $3Scan(w)_j$ respectively, we have $s < u < t$ implying:

$$VN[j,k]_s = VN[j,k]_u = VN[j,k]_t = VN[j,k]_W$$

$$OVN[j,k]_s = OVN[j,k]_u = OVN[j,k]_t = OVN[j,k]_W$$

$$PVN[j,k]_s = PVN[j,k]_u = PVN[j,k]_t = PVN[j,k]_W$$

for all writers k as desired. □

This result permits us to think of the values of the VN's, OVN's, and PVN's observed by a write W, those values on the basis of which W chooses the VN's, OVN's, and PVN's that it writes, to have been read by an atomic scan of all the writers' registers acting at the point $Scan(W)$. This meaning of the $Scan(W)$ action is fundamental to the remainder of the proof.

Now that we have established the meaning of the $Scan(W)$ action, we will present two theorems that capture the essence of the relative meanings of the VN's, OVN's, and PVN's. The first of these theorems states that for given writers i and j, if writer i "sees" writer j's version number at a given point, if $OVN[i,j] = VN[j,i]$ at that point, then writer i has both scanned and written since the last write by writer j. The second theorem states that for given writers i and j, if writer i sees neither writer j's VN nor writer j's PVN at a given point, if $OVN[i,j] \neq VN[j,i]$ and $OVN[i,j] \neq PVN[j,i]$ at that point, then writer j completed two writes between the scan and write actions of the most recent write completed by writer i. Let us first prove a little lemma.

Lemma 6 *Let s be any state in an execution of the composition automaton. Let i be any writer and let W_i be the last write by writer i for which $Write(W_i) < s$. Let j be any writer for which there exists a write W_j such that $Scan(W_i) < Write(W_j) < s$. Let t be the state following $Write(W_j)$. Then $OVN[i,j]_s \neq VN[j,i]_t$.*

Proof of Lemma 6: Let j, W_j, and t be as in the lemma statement. Let u and v be the states following $Scan(W_j)$ and $PScan(W_j)_i$ respectively. Then there are four cases we must consider:

Case 1: $v < Scan(W_i)$. Then since we have $u < PScan(W_j)_i < v$, $u < Scan(W_i) < Write(W_j)$. Since writer j is in the process of performing the write W_j between u and $Write(W_j)$, ie. since $Start(W_j) < u < Write(W_j) < Finish(W_j)$, there are no other writes W_j' by writer j for which $u < Write(W_j') < Write(W_j)$ and consequently $VN[j,i]_{s'}$ is constant for all s', $u \leq s' < Write(W_j)$. In particular, if x is the state following $Scan(W_i)$ then:

$$VN[j,i]_x = VN[j,i]_u.$$

Let y be the state following $Write(W_i)$. Then by Lemma 1 we have:

$$OVN[i,j]_y = VN[j,i]_x$$

and

$$VN[j,i]_t \neq VN[j,i]_u.$$

By choice of W_i and hence of y, $OVN[i,j]$ remains constant between y and s. Consequently:

$$OVN[i,j]_s = OVN[i,j]_y.$$

Putting the above equations together yields:

$$OVN[i,j]_s = OVN[i,j]_y = VN[j,i]_x = VN[j,i]_u \neq VN[j,i]_t$$

as desired.

Case 2: $Scan(W_i) < v < Write(W_i)$. Now $PreOVN[i,j]$ remains constant between $PWrite(W_i)_j$ and $Write(W_i)$ and by Lemma 1 equals $OVN[i,j]_y$ if y is the state following $Write(W_i)$. Since $PWrite(W_i)_j < Scan(W_i) < v < Write(W_i)$ we thus have:

$$PreOVN[i,j]_v = OVN[i,j]_y.$$

By Lemma 1, we have:

$$VN[j,i]_t \neq PreOVN[i,j]_v.$$

By choice of W_i and thus of y, $OVN[i,j]$ remains constant between y and s. Thus:

$$OVN[i,j]_s = OVN[i,j]_y.$$

Putting the above equations together yields:

$$OVN[i,j]_s = OVN[i,j]_y = PreOVN[i,j]_v \neq VN[j,i]_t$$

as desired.

Case 3: $Write(W_i) < v$ but $u < Write(W_i)$. This implies $2Scan(W_j)_i < u < Write(W_i) < v < 3Scan(W_j)_i$. By Lemma 4 this is impossible.

Case 4: $Write(W_i) < v$ and $Write(W_i) < u$. Note that $u < v < Write(W_j) < s$. Now by choice of W_i, $OVN[i,j]$ equals the constant $OVN[i,j]_s$ between $Write(W_i)$ and s. In particular:

$$OVN[i,j]_u = OVN[i,j]_s.$$

Now by Lemma 1:

$$VN[j,i]_t \neq OVN[i,j]_u.$$

Putting these equations together yields:

$$OVN[i,j]_s = OVN[i,j]_u \neq VN[j,i]_t$$

as desired.

This completes proof of Lemma 6. □

Theorem 7 *Let i and j be writes, $i \neq j$. Let s be any state in an execution of the composition automaton. Let W_i and W_j be the most recent writes by writers i and j for which $Write(W_i) < s$ and $Write(W_j) < s$. Then $OVN[i,j]_s = VN[j,i]_s$ if and only if $Write(W_j) < Scan(W_i)$.*

Proof of Theorem 7: Let us first show that:

$$OVN[i,j]_s = VN[j,i]_s \Longrightarrow Write(W_j) < Scan(W_i).$$

Assume otherwise, that $OVN[i,j]_s = VN[j,i]_s$ but that $Scan(W_i) < Write(W_j)$. Let v be the state following $Write(W_j)$. Then by choice of W_j we have $Scan(W_i) < Write(W_j) < s$ implying by Lemma 6 that:

$$OVN[i,j]_s \neq VN[j,i]_v.$$

Since by choice, W_j is the last write by writer j such that $Write(W_j) < s$, the value of $VN[j,i]$ remains constant between v and s implying that:

$$VN[j,i]_v = VN[j,i]_s.$$

Putting these together yields

$$OVN[j,i]_s \neq VN[j,i]_v = VN[j,i]_s$$

which contradicts our initial assumption that $OVN[i,j]_s = VN[j,i]_s$. Thus the first direction of the theorem is proved.

Now, let us show that:

$$Write(W_j) < Scan(W_i) \Longrightarrow OVN[i,j]_s = VN[j,i]_s.$$

Assume $Write(W_j) < Scan(W_i)$. Since W_j is the last write by writer j such that $Write(W_j) < s$, $VN[j,i]_{s'} = VN[j,i]_s$ for all states s' such that $Write(W_j) < s' < s$. In particular, if t is the state following $Scan(W_i)$, then since by assumption $Write(W_j) < Scan(W_i) < s$, we have $Write(W_j) < t < s$ implying $VN[j,i]_t = VN[j,i]_s$. By Lemma 1, $OVN[i,j]_s = VN[j,i]_t$ and thus $OVN[i,j]_s = VN[j,i]_s$ as desired. This concludes the proof of Theorem 7. □

Theorem 8 *Let i be any writer and s be any state in an execution of the composition automaton. Let W_i be the last write by writer i such that $Write(W_i) < s$. Then $N(i)_s = 0$ if and only if there is a writer $j \neq i$ that made writes W_j and W'_j, $W_j \neq W'_j$ such that*

$$Scan(W_i) < Write(W'_j) < Write(W_j) < s.$$

Proof of Theorem 8: Assume there exist two writes W'_j and W_j by writer j such that $Scan(W_i) < Write(W'_j) < Write(W_j) < s$; let W'_j and W_j be the last such writes. Let t and u be the states following $Write(W'_j)$ and $Write(W_j)$ respectively. Then by Lemma 6 we have:

$$OVN[i,j]_s \neq VN[j,i]_t$$

and

$$OVN[i,j]_s \neq VN[j,i]_u.$$

By choice, W'_j is the last write by writer j such that $Write(W'_j) < Write(W_j)$, thus if v is the state following $Scan(W_j)$, we have $VN[j,i]_v = VN[j,i]_t$. By Lemma 1 we have $PVN[j,i]_u = VN[j,i]_v$, thus:

$$PVN[j,i]_u = VN[j,i]_t.$$

Now by choice, W_j is the last write by writer j such that $Write(W_j) < s$, thus:

$$VN[j,i]_s = VN[j,i]_u$$

and

$$PVN[j,i]_s = PVN[j,i]_u.$$

Putting the above equations together we get:

$$OVN[i,j]_s \neq VN[j,i]_u = VN[j,i]_s$$

and

$$OVN[i,j]_s \neq VN[j,i]_t = PVN[j,i]_u = PVN[j,i]_s.$$

Consequently, $N(i)_s = 0$. Thus if j, W_j', and W_j exist as in the theorem statement, then $N(i)_s = 0$.

Now for the other direction. Assume $N(i)_s = 0$. This means $PVN[j,i]_s \neq OVN[i,j]_s$ and $VN[j,i]_s \neq OVN[i,j]_s$ for some writer j. We have three cases:

1. There are no writes W_j by writer j for which $Scan(W_i) < Write(W_j) < s$. Let t be the state following $Scan(W_i)$. Then $VN[j,i]$ remains constant between t and s implying $VN[j,i]_s = VN[j,i]_t$. By Lemma 1, $VN[j,i]_t = OVN[i,j]_s$ and we have:

   $$VN[j,i]_s = VN[j,i]_t = OVN[i,j]_s.$$

   Thus this case is not possible.

2. There is exactly one write W_j by writer j for which $Scan(W_i) < Write(W_j) < s$. Let t and x be the states following $Scan(W_i)$ and $Write(W_j)$ respectively. Then

   $$PVN[j,i]_s = PVN[j,i]_x = VN[j,i]_t = OVN[i,j]_s.$$

   Thus this case is not possible.

3. There are at least two writes W_j by writer j for which $Scan(W_i) < Write(W_j) < s$. This implies the existence of W_j and W_j' as required by the theorem statement.

Thus $N(i) = 0$ implies there exists a writer j and writes W_j and W_j' by writer j such that $Scan(W_i) < Write(W_j') < Write(W_j) < s$. This completes the proof of the theorem. □

We will now apply the two theorems that we have just proved to prove several useful and interesting facts about some of the various constructs, such as $VNS(i)_s$, $N(i)_s$, and $F(s)$, that we defined earlier. The first of these facts, expressed in the following Lemma, shows that for any state s and any writers i and j, if $VNS(i)_s \neq VNS(j)_s$ then one of $VNS(i)_s$ and $VNS(j)_s$ is a proper subset of the other.

Lemma 9 *Let i and j be writers and s be a state in an execution of the composition automaton. If $VNS(i)_s \setminus VNS(j)_s \neq \emptyset$ then $VNS(j)_s$ is a proper subset of $VNS(i)_s$.*

Proof of Lemma 9: Given $VNS(i)_s \setminus VNS(j)_s \neq \emptyset$, let $k \in VNS(i)_s \setminus VNS(j)_s$. Let W_i, W_j and W_k be the last writes by writers i, j, and k respectively for which $Write(W_i) < s$, $Write(W_j) < s$, and $Write(W_k) < s$. Since $k \in VNS(i)_s$, $VN[k,i]_s = OVN[i,k]_s$ which by Theorem 7 implies $Write(W_k) < Scan(W_i)$. Also, since $k \notin VNS(j)_s$, $VN[k,j]_s \neq OVN[j,k]_s$ implying by Theorem 7 that $Scan(W_j) < Write(W_k)$. This implies $Scan(W_j) < Scan(W_i)$. Now by symmetry, $VNS(j)_s \setminus VNS(i)_s \neq \emptyset$ would imply $Scan(W_i) < Scan(W_j)$, thus we know $VNS(j)_s \setminus VNS(i)_s = \emptyset$. This implies $VNS(j)_s \subset VNS(i)_s$. Since $k \in VNS(i)_s \setminus VNS(j)_s$, the inclusion is proper and the lemma is proved. □

Corollary 10 *Let i and j be writers and s be a state in an execution of the composition automaton. Then:*

1. *$VNS(j)_s$ is a proper subset of $VNS(i)_s$ if and only if $|VNS(j)_s| < |VNS(i)_s|$.*
2. *$VNS(j)_s = VNS(i)_s$ if and only if $|VNS(j)_s| = |VNS(i)_s|$.*

Proof of Corollary 10: This follows directly from Lemma 9 and elementary set theory. □

The following lemma presents another important fact. It is important because it and the corollary that follows it relate the two principal values that are used for determining the value of $F(s)$ at a state s, namely the $|VNS(i)_s|$ and the $N(i)_s$.

Lemma 11 *Let i and j be any writers, $i \neq j$, and let s be any state in an execution of the composition automaton. Then:*

$$|VNS(i)_s| > |VNS(j)_s| \Longrightarrow N(i)_s \geq N(j)_s.$$

Proof of Lemma 11: Assume otherwise, that $|VNS(i)_s| > |VNS(j)_s|$ but $N(i)_s < N(j)s$. By Corollary 10, $VNS(j)_s$ is a proper subset of $VNS(i)_s$ implying that there is some $k \in VNS(i)_s \setminus VNS(j)_s$. By definition of the VNS this means that $VN[k,i]_s = OVN[i,k]_s$ but $VN[k,j]_s \neq OVN[j,k]_s$. Let W_i, W_j, and W_k be the last writes by writers i, j, and k respectively for which $Write(W_i) < s$, $Write(W_j) < s$, and $Write(W_k) < s$. Then by Theorem 7 we have $Scan(W_j) < Write(W_k)$ but $Write(W_k) < Scan(W_i)$ and thus $Scan(W_j) < Scan(W_i)$. Now $N(i)_s < N(j)_s$ implies $N(i)_s = 0$ and $N(j)_s = 1$. By Theorem 8, $N(i)_s = 0$ implies that there exists some writer l and two writes W_l and W_l' such that:

$$Scan(W_i) < Write(W_l') < Write(W_l) < s.$$

But $Scan(W_j) < Scan(W_i)$ implies that:

$$Scan(W_j) < Write(W_l') < Write(W_l) < s.$$

By Theorem 8 again, we have $N(j)_s = 0$ contradicting the above. Thus our assumption is incorrect and the lemma is proved. □

Corollary 12 *Let i and j be any writers $i \neq j$, and let s be any state in an execution of the composition automaton. Then:*

1. $|VNS(i)_s| > |VNS(j)_s| \Longrightarrow |VNS(i)_s| + N(i)_s > |VNS(j)_s| + N(j)_s$
2. $|VNS(i)_s| + N(i)_s > |VNS(j)_s| + N(j)_s \Longrightarrow |VNS(i)_s| \geq |VNS(j)_s|$
3. $|VNS(i)_s| + N(i)_s > |VNS(j)_s| + N(j)_s \Longrightarrow N(i)_s \geq N(j)_s$
4. $|VNS(i)_s| + N(i)_s = |VNS(j)_s| + N(j)_s \Longrightarrow |VNS(i)_s| = |VNS(j)_s|$
5. $|VNS(i)_s| + N(i)_s = |VNS(j)_s| + N(j)_s \Longrightarrow N(i)_s = N(j)_s$

Proof of Corollary 12: All parts follow directly from Lemma 11. □

Corollary 13 *Let s be any state in an execution of the composition automaton. Then:*

$$VNS(i)_s \subset VNS(F(s))_s$$

for all writers i.

Proof of Corollary 13: Assume otherwise. Then for some $i \neq F(s)$,

$$VNS(i)_s \setminus VNS(F(s))_s \neq \emptyset.$$

Then by Lemma 9, $VNS(F(s))_s$ is a proper subset of $VNS(i)_s$. Then

$$|VNS(F(s))_s| < |VNS(i)_s|$$

implying by Corollary 12 that

$$|VNS(F(s))_s| + N(F(s))_s < |VNS(i)_s| + N(i)_s$$

contradicting the definition of F(s). Thus our assumption is incorrect and the corollary holds. □

The following lemma and corollary demonstrate that at each step s, the function N takes on a non-zero value for at least one writer, and in particular, $N(F(s))_s = 1$.

Lemma 14 *Let s be any state in an execution of the composition register. Then there exists some writer i for which $N(i)_s = 1$.*

Proof of Lemma 14: Of all the writes W, by any writer, for which $Write(W) < s$, let W_i be the one for which $Scan(W_i)$ most recently preceeds s. Let i be the writer that performed the write W_i. Assume $N(i)_s = 0$. Then by Theorem 8 there exists a writer j and writes W_j and W_j' by writer j for which

$$Scan(W_i) < Write(W_j') < Write(W_j) < s.$$

But W_j must have begun after W_j' finished implying

$$Write(W_j') < Scan(W_j) < Write(W_j).$$

Consequently,

$$Scan(W_i) < Scan(W_j) < Write(W_j) < s$$

contradicting our choice of W_i. Thus our assumption is incorrect and $N(i)_s = 1$ proving the lemma. □

Corollary 15 *Let s be any state in an execution of the composition register. Then we have $N(F(s))_s = 1$.*

Proof of Corollary 15: Let i be some writer such that $N(i)_s = 1$; such a writer exists by Lemma 14. If $i = F(s)$ then we're done. Otherwise we have three cases:

1. $|VNS(F(s))_s| + N(F(s))_s > |VNS(i)_s| + N(i)_s$. By Corollary 12, $N(F(s))_s \geq N(i)_s = 1$ and we're done.
2. $|VNS(F(s))_s| + N(F(s))_s = |VNS(i)_s| + N(i)_s$. By Corollary 12, $N(F(s))_s = N(i)_s = 1$ and we're done.
3. $|VNS(F(s))_s| + N(F(s))_s < |VNS(i)_s| + N(i)_s$. This case cannot occur as it would contradict the definition of F(s).

This completes the proof of the corollary. □

13 Placement of Writes

We will now use the facts we have established to prove two theorems that are the basis for the placement of atomic write points in an execution of the composition automaton. First, however, we will need the following definition.

DEFINITION: Let W be a write by writer i that does not time out. Let s be the state following $Write(W)$. We will call the write W *potent* if $F(s) = i$. We will call the write W *impotent* if $F(s) \neq i$.

The first of the two theorems we will now prove states that if W is an impotent write, then F has the same values for the states immediately preceeding and following $Write(W)$. Intuitively, this is very desirable behavior. If a writer writes a new value V to its register, one would expect that in doing so, it would either change the value of the composition register to V, or it would leave the value in the composition register unchanged. It would be highly undesirable if writes could cause a value that had previously been current, but had since been overwritten, to become current again.

The second of the two theorems that we are about to prove states that if W is any impotent write, then there is some potent write W' such that W' wrote its value and new VN, OVN, and PVN numbers between the scan and write actions of W. This, again, is what one would expect. A writer performing its scan and write operations during an interval in which no other writes are occuring should change the value of the composition register to that of its own register when it completes its write. These two theorems provide us with points at which to insert an "atomic" action for both potent and impotent writes, as we will see later.

Theorem 16 *Let W be an impotent write written by writer i. Let s' and s be the states preceeding and following $Write(W)$ respectively. Then $F(s') = F(s)$.*

Proof of Theorem 16: We will first prove a few propositions that will be useful in the proof of the theorem. In all of these propositions, we will assume W, i, s', and s are as above. Note that $i \neq F(s)$ since W is impotent.

Proposition 16.1 $i \in VNS(F(s))_{s'}$.

Proof of Proposition 16.1: Assume otherwise. Then

$$OVN[F(s), i]_{s'} \neq VN[i, F(s)]_{s'}$$

implying by Theorem 7 that if $W_{F(s)}$ is the last write by writer $F(s)$ for which we have $Write(W_{F(s)}) < s'$ then there is some write W' by writer i such that

$$Scan(W_{F(s)}) < Write(W') < s'.$$

Then since $W_{F(s)}$ is also the last write by writer $F(s)$ for which $Write(W_{F(s)}) < s$ and

$$Scan(W_{F(s)}) < Write(W') < s' < Write(W) < s$$

Theorem 8 tells us that $N(F(s))_s = 0$ contradicting Corollary 15. Thus the proposition holds. □

Proposition 16.2 $F(s') \neq i$.

Proof of Proposition 16.2: By Corollary 13 we know that $VNS(F(s))_{s'} \subset VNS(F(s'))_{s'}$ and by the above, $i \in VNS(F(s))_{s'}$ thus $i \in VNS(F(s'))_{s'}$. Now by Lemma 2 we know $i \notin VNS(i)_{s'}$. We conclude $F(s') \neq i$. □

Proposition 16.3 *For all writers j, $j \neq i$, $VNS(j)_s = VNS(j)_{s'} \setminus \{i\}$.*

Proof of Proposition 16.3: Let j be a writer, $j \neq i$. Since there are no writes W_k by any writer $k \neq i$ such that $s' < Write(W_k) < s$, we know that $VN[k,j]_s = OVN[j,k]_s$ if and only if $VN[k,j]_{s'} = OVN[j,k]_{s'}$ for all writers k, $k \neq i$. Thus we have $k \in VNS(j)_s$ if and only if $k \in VNS(j)_{s'}$ for $k \neq i$.

If we had $i \in VNS(j)_s$ then by Theorem 7 we would have $s' < Write(W) < Scan(W_j) < s$ where W_j is the last write by writer j for which $Write(W_j) < s$; this would clearly contradict our choice of s' and s which are chosen such that $Write(W)$ is the only action between them. Therefore, $i \notin VNS(j)_s$.

Thus we have $k \in VNS(j)_s$ if and only if $k \in VNS(j)_{s'}$ for $k \neq i$, and $i \notin VNS(j)_s$. By elementary set theory, we conclude $VNS(j)_s = VNS(j)_{s'} \setminus \{i\}$. Since j is an arbitrary writer, our proof of the Proposition 16.3 is complete. □

Proposition 16.4

$$|VNS(F(s'))_s| = |VNS(F(s'))_s| - 1 \quad and \quad |VNS(F(s))_s| = |VNS(F(s))_s| - 1.$$

Proof of Proposition 16.4: As was noted in the proof of Proposition 16.2, $i \in VNS(F(s))_{s'}$ and $i \in VNS(F(s'))_{s'}$. By Proposition 16.2, $F(s') \neq i$, and $F(s) \neq i$ because W is impotent. The proposition thus follows from Proposition 16.3 and elementary set theory. □

Proposition 16.5 *Let j be any writer for which $i \in VNS(j)_{s'}$. Then $N(j)_s = N(j)_{s'}$.*

Proof of Proposition 16.5: By definition, $i \in VNS(j)_{s'}$ implies $VN[i,j]_{s'} = OVN[j,i]_{s'}$. By Lemma 1 we have $PVN[i,j]_s = VN[i,j]_{s'}$ and thus $PVN[i,j]_s = VN[i,j]_{s'} = OVN[i,j]_{s'} = OVN[j,i]_s$. Thus $PVN[i,j]_s = OVN[j,i]_s$. By definition, $N(j)_s = 0$ if and only if there exists some writer k such that $VN[k,j]_s \neq OVN[j,k]_s$ and $PVN[k,j]_s \neq OVN[j,k]_s$. Since $PVN[i,j]_s = OVN[j,i]_s$, there exists such a k if and only if there exists such a k, $k \neq i$. Since $j \neq i$, $OVN[j,l]_{s'} = OVN[j,l]_s$ for all l, $l \neq i$; also, $VN[l,j]_{s'} = VN[l,j]_s$ and $PVN[l,j]_{s'} = PVN[l,j]_s$ for all l, $l \neq i$. This implies that there exists such a $k \neq i$ if and only if $VN[k,j]_{s'} \neq OVN[j,k]_{s'}$ and $PVN[k,j]_{s'} \neq OVN[j,k]_{s'}$. But by definition, $N(j)_{s'} = 0$ if and only if either such a $k \neq i$ exists or if $VN[i,j]_{s'} \neq OVN[j,i]_{s'}$ and $PVN[i,j]_{s'} \neq OVN[j,i]_{s'}$. We have seen that $VN[i,j]_{s'} = OVN[j,i]_{s'}$ and we thus conclude that $N(j)_s = 0$ if and only if $N(j)_{s'} = 0$. Since N takes on only the values 1 and 0, our proof is complete. □

Proposition 16.6 $N(F(s))_s = N(F(s))_{s'} and N(F(s'))_s = N(F(s'))_{s'}$.

Proof of Proposition 16.6: As was noted in the proof of Proposition 16.2, $i \in VNS(F(s))_{s'}$ and $i \in VNS(F(s'))_{s'}$. The proposition follows immediately from Proposition 16.5. □

We now proceed with the proof of Theorem 16. Assume that $F(s') \neq F(s)$; we will derive a contradiction. Now by definition of $F(s')$, one of two cases must occur:

1. $|VNS(F(s'))_{s'}| + N(F(s'))_{s'} > |VNS(F(s))_{s'}| + N(F(s))_{s'}$. Then by Propositions 16.4 and 16.6,

$$\begin{aligned} |VNS(F(s'))_s| + N(F(s'))_s &= |VNS(F(s'))_{s'}| + N(F(s'))_{s'} - 1 \\ &> |VNS(F(s))_{s'}| + N(F(s))_{s'} - 1 = \\ &\quad |VNS(F(s))_s| + N(F(s))_s \end{aligned}$$

Thus $|VNS(F(s'))_s| + N(F(s'))_s > |VNS(F(s))_s| + N(F(s))_s$ contradicting the definition of $F(s)$.

2. $|VNS(F(s'))_{s'}| + N(F(s'))_{s'} = |VNS(F(s))_{s'}| + N(F(s))_{s'}$ and $F(s') > F(s)$. Then by Propositions 16.4 and 16.6,

$$\begin{aligned} |VNS(F(s'))_s| + N(F(s'))_s &= |VNS(F(s'))_{s'}| + N(F(s'))_{s'} - 1 \\ &= |VNS(F(s))_{s'}| + N(F(s))_{s'} - 1 \\ &= |VNS(F(s))_s| + N(F(s))_s \end{aligned}$$

Thus $|VNS(F(s'))_s| + N(F(s'))_s = |VNS(F(s))_s| + N(F(s))_s$ and $F(s') > F(s)$ contradicting the definition of $F(s)$.

Thus our assumption is incorrect and $F(s') = F(s)$ as desired. This completes the proof of Theorem 16. □

Corollary 17 *F remains constant between consecutive $Write(W)$ actions for potent writes W.*

Proof of Corollary 17: We noted earlier that the only points at which the values of $VN[i,j]$, $OVN[i,j]$, and $PVN[i,j]$ may change are at the $Write(W)$ actions for writes W by writer i. Formally, if A is an action in an execution of the composition automaton and if A is not equal to $Write(W)$ for any write W, and if s' and s are the states preceeding and following A respectively, then:

$$\begin{aligned} VN[i,j]_{s'} &= VN[i,j]_s \\ PVN[i,j]_{s'} &= PVN[i,j]_s \\ OVN[i,j]_{s'} &= OVN[i,j]_s \end{aligned}$$

for all writers i and j. Consequently, $F(s') = F(s)$. Theorem 16 implies that $F(s') = F(s)$ even if $A = Write(W)$ for an impotent write W. Since $Write(W)$ actions are associated only with potent and impotent writes W, the correctness of the corollary follows. □

Theorem 18 *Let i be any writer and W_i be any impotent write by writer i. Then there exists some writer j, $j \neq i$ and some write W_j by writer j such that $Scan(W_i) < Write(W_j) < Write(W_i)$.*

Proof of Theorem 18: Let s be the state immediately following $Write(W_i)$. Then W_i is the last write by writer i for which $Write(W_i) < s$. Let $j = F(s)$. Note $j \neq i$ because W_i is impotent. Since, by Corollary 17, the value of F remains constant between potent writes, we have $j = F(s')$ where s' is the state following the last potent write W_j for which $Write(W_j) < s$. Now W_j is clearly written by writer j as $F(s') = j$ and W_j is potent. Because F equals j between s' and s, we know by definition of an impotent write that there can be no impotent writes W_j' by writer j for which $s' < Write(W_j') < s$. Also, because W_j is the most recent potent write before s, we know that there can be no potent writes W_j' by writer j for which $s' < Write(W_j') < s$. Therefore W_j is the last write by writer j for which $Write(W_j) < s$.

Assume now that there is no potent write W for which $Scan(W_i) < Write(W) < Write(W_i)$. Then, in particular, $Write(W_j) < Scan(W_i)$. By Theorem 7 this implies that $OVN[i,j]_s = VN[j,i]_s$. Thus $j \in VNS(i)_s \setminus VNS(j)_s$ and thus by Lemma 9, $VNS(j)_s$ is a proper subset of $VNS(i)_s$. By Corollary 12 we have $|VNS(i)_s| + N(i)_s > |VNS(j)_s| + N(j)_s$. This implies, by definition of $F(s)$ that $F(s)$ could not possibly equal j. Thus our assumption is incorrect and there is a writer j, $j \neq i$, and a potent write W_j by writer j for which $Scan(W_i) < Write(W_j) < Write(W_i)$. This completes the proof of Theorem 18. □

We are now ready to place $Atomic(W)$ action for each write W.

1. For each potent write W, define the internal action $Atomic(W)$ to equal $Write(W)$. Clearly, $Start(W) < Atomic(W) < Finish(W)$.

2. For each impotent write W, we know by Theorem 18 that there exists some potent write W' such that $Scan(W) < Write(W') < Write(W)$; let W' be the last such potent write. Insert an action $Atomic(W)$ immediately preceeding $Write(W')$. Again, since we are inserting $Atomic(W)$ between $Scan(W)$ and $Write(W)$, it is clear that $Start(W) < Atomic(W) < Finish(W)$.

   Note that we may have to insert several *Atomic* actions for impotent writes immeditately preceeding a single potent write W'. This is not a problem; since we have only m writers, there are at most $m - 1$ writers that could be performing impotent writes at the point $Write(W')$. We are thus inserting a finite number of actions before any $Write(W')$.

3. For each write W that times out, we know from the fact that it timed out that, for some writer i, W saw the contents of writer i's register change twice. Since the values in writer i's register that are compared between scans (the $VN[i,j]$, $OVN[i,j]$, $PVN[i,j]$, and $Value[i]$) change only at the points $Write(W')$ for writes W' by writer i that do not time out, the two observed changes must have been caused by separate writes by writer i. The second of these writes, call it W', must have begun after the first finished. Thus we have $Start(W) < Scan(W') < Write(W') < Finish(W)$. Whether W' is potent or impotent, we have $Scan(W') < Atomic(W') \leq Write(W')$, thus if we insert $Atomic(W)$

immediately preceeding $Write(W')$ it is clear that we will have $Start(W) < Atomic(W) < Finish(W)$.

Here, as was the case with impotent writes, we may have to insert several *Atomic* actions immediately before a given *Write* action; here, as before, this causes no problem.

Before we continue, there are a few things that we should note about our placement of the *Atomic* actions for writes. First, for every write W that does not time out, we have $Scan(W) < Atomic(W) \leq Write(W)$. Second, if e is an execution of the composition automaton in which no *Atomic* actions have been inserted and s is a state in e, then once the *Atomic* actions for writes have been inserted into e to yield e', the most recent *Atomic* write action preceeding s in e' is that of a potent write. Third, from Corollary 17 we see that the value of F remains constant between consecutive *Atomic* actions of writes.

14 Placement of Reads

Now that all of the writes have been placed, we need to show that reads will behave in the desired manner. This is demonstrated by the following theorem that, although it is not constructive and does not tell us exactly where to place the "atomic" action associated with a read, tells us that all reads that do not time out do indeed return legitimate values.

Theorem 19 *Let R be any read that did not time out. Let i be the number of the writer whose value was chosen to be returned by R; $i = F(R)$. Let W be the last write by writer i for which $Write(W) < 3Scan(R)_i$. Then the following hold.*

1. $Value(R) = Value(W)$.
2. $Atomic(W) < Finish(R)$.
3. *There does not exist a write W' for which $Atomic(W) < Atomic(W') < Start(R)$.*

Proof of Theorem 19: We will prove the parts separately. Assume R, W, and i are as defined above.

1. Since W is the last write by writer i for which $Write(W) < 3Scan(R)_i$, and R returns the value read by $3Scan(R)_i$ from writer i's register, R returns the value written by W.
2. Note that by the way we placed $Atomic(W')$ actions for writes W', $Atomic(W') \leq Write(W')$ for all writes W'. By choice of W, $Write(W) < 3Scan(R)_i$. By definition, of $Finish(R)$, $3Scan(R)_i \leq Finish(R)$. We conclude that $Atomic(W) < Finish(R)$.

3. This is the hard part. We will derive a contradiction after demonstrating the following sequence of propositions. Thus the first step of our proof is to assume the negation of what we are trying to prove. Namely, assume that there exists some write W' such that $Atomic(W) < Atomic(W') < Start(R)$. Note that all of the following propositions are dependent upon the existence of W' and that all assume R, W, and i to be defined as above.

Proposition 19.1 *There is no write W'' by writer i for which*

$$1Scan(R)_i < Write(W'') < 3Scan(R)_i.$$

Consequently,

$$\begin{aligned} VN[i,j]_s &= VN[i,j]_R \\ OVN[i,j]_s &= OVN[i,j]_R \\ PVN[i,j]_s &= PVN[i,j]_R \end{aligned}$$

for all states s, $1Scan(R)_i < s < 3Scan(R)_i$ and all writers j. Also, W is the last write by writer i for which $Write(W) < s$ for all states s, $1Scan(R)_i < s < 3Scan(R)_i$.

Proof of Proposition 19.1: Let t and u be the states following $1Scan(R)_i$ and $3Scan(R)_i$ respectively. Since the last three scans made by R see the same values, we have $VN[i,i]_t = VN[i,i]_u$. Assume there exists some write W'' by writer i such that $1Scan(R)_i < Write(W'') < 3Scan(R)_i$. Then by Lemma 3 there exists some write W''' by writer i for which $t < Scan(W''') < Write(W''') < u$; let W''' be the last such write. Then by the way we placed the *Atomic* actions for writes, we have $Scan(W''') < Atomic(W''') < Write(W''')$. Since we have just chosen W''' to be the last write by writer i for which $Write(W''') < u$, W''' must also be the last write by writer i for which $Write(W''') < 3Scan(R)_i$. Then by choice of W, we have $W = W'''$. But we have assumed

$$Atomic(W) < Start(R)$$

while

$$Start(R) < 1Scan(R) < t < Scan(W''') < Atomic(W''').$$

This contradiction implies that our assumption is incorrect and the proposition is proved. □

Proposition 19.2 $Scan(W) < Start(R)$.

Proof of Proposition 19.2: By assumption, there exists some write W' for which $Atomic(W) < Atomic(W') < Start(R)$, thus $Atomic(W) < Start(R)$. Now by the way we placed the *Atomic* actions for writes, $Scan(W) < Atomic(W) \leq Write(W)$. Thus we have $Scan(W) < Atomic(W) < Start(R)$ as desired. □

Proposition 19.3 $i \notin VNS(i)_R$.

Proof of Proposition 19.3: Let s be the state following $1Scan(R)_i$. Then $OVN[i,i]_s = OVN[i,i]_R$ and $VN[i,i]_s = VN[i,i]_R$. Thus, since Lemma 2 implies $OVN[i,i]_s \neq VN[i,i]_s$, we have $OVN[i,i]_R \neq VN[i,i]_R$. Hence $i \notin VNS(i)_R$ as desired. □

Proposition 19.1 showed that writer i is incapable of performing the $Write$ actions of any writes between $1Scan(R)_i$ and $3Scan(R)_i$. Since the principal values in writer i's register (the $VN[i,j]$, $OVN[i,j]$, and $PVN[i,j]$) thus remain constant between $1Scan(R)_i$ and $3Scan(R)_i$, the interval from $1Scan(R)_i$ to $3Scan(R)_i$ forms a sort of "magic interval" in which we can infer many things about the behavior of other writers. The following inequalities are particularly important in this respect:

$$1Scan(R)_i < 2Scan(R)_j < 3Scan(R)_j < 3Scan(R)_i$$

for all writers j, $j < i$, and

$$1Scan(R)_i < 1Scan(R)_j < 2Scan(R)_j < 3Scan(R)_i$$

for all writers j, $j > i$. These inequalities are fundamental because they define intervals, defined in terms of reads of writer j's register, that are contained within the interval from $1Scan(R)_i$ to $3Scan(R)_i$. Since these inequalities are fundamental to the proof of the remaining propositions, they will have the undesirable effect of introducing a division into the cases of $j < i$ and $j > i$ in all of the following propositions.

Proposition 19.4 *(a) Let j be the number of any writer $j < i$. If $j \in VNS(i)_R$ then there is no write W_j by writer j such that $Scan(W) < Write(W_j) < 3Scan(R)_j$.*

(b) Let j be the number of any writer $i < j$. If $j \in VNS(i)_R$ then there is no write W_j by writer j such that $Scan(W) < Write(W_j) < 2Scan(R)_j$.

Proof of Proposition 19.4:

(a) Assume otherwise, that there is some writer j, $j < i$, $j \in VNS(i)_R$ that performed a write W_j such that:

$$Scan(W) < Write(W_j) < 3Scan(R)_j$$

and let W_j be the last such write. Let s and t be the states following $3Scan(R)_j$ and $Write(W_j)$ respectively. By Proposition 19.1, W is the last write by writer i such that $Write(W) < s$. Then by Lemma 6 we have:

$$OVN[i,j]_s \neq VN[j,i]_t.$$

Since W_j is the last write by writer j such that $Write(W_j) < 3Scan(R)_j$, $VN[j,i]$ remains constant between $Write(W_j)$ and $3Scan(R)_j$; in particular,

$$VN[j,i]_t = VN[j,i]_R.$$

By Proposition 19.1, since $1Scan(R)_i < s < 3Scan(R)_i$, we have:

$$OVN[i,j]_R = OVN[i,j]_s.$$

Putting these equations together yields:

$$OVN[i,j]_R = OVN[i,j]_s \neq VN[j,i]_t = VN[j,i]_R$$

contradicting our assumption that $j \in VNS(i)_R$. Thus our assumption is incorrect and the first half of the proposition is proved.

(b) The second part of the proof of the proposition follows exactly like the first; $1Scan(R)_j$ replaces $2Scan(R)_j$, and $2Scan(R)_j$ replaces $3Scan(R)_j$.

This completes the proof of Proposition 19.4. □

Proposition 19.5 *Let j be any writer. If $i \in VNS(j)_R$ then $VNS(i)_R$ is a proper subset of $VNS(j)_R$.*

Proof of Proposition 19.5:

(a) Case 1: $j < i$. Since $i \in VNS(j)_R$ we have $OVN[j,i]_R = VN[i,j]_R$. Let W_j be the last write by writer j for which $Write(W_j) < 2Scan(R)_j$. Let s be the state following $2Scan(R)_j$. By Proposition 19.1, $VN[i,j]_s = VN[i,j]_R$. By choice of s, $OVN[j,i]_s = OVN[j,i]_R$ and thus $OVN[j,i]_s = VN[i,j]_s$. By Proposition 19.1 and choice of W, W is the last write by writer i for which $Write(W) < s$. By choice of W_j, W_j is the last write by writer j for which $Write(W_j) < s$. Then by Theorem 7, $Write(W) < Scan(W_j)$. This, of course, implies $Scan(W) < Scan(W_j)$.

Let k be any writer for which $k \in VNS(i)_r$. Note then that by Proposition 19.3, $k \neq i$. Let W_k be the last write by writer k for which $Write(W_k) < Scan(W)$. Then by Proposition 19.4, W_k is also the last write by writer k for which $Write(W_k) < 2Scan(R)_j$ since $2Scan(R)_j < 2Scan(R)_k$ for $k > i > j$, and $2Scan(R)_j < 3Scan(R)_k$ if $k < i$. Thus W_k is the last write by writer k for which $Write(W_k) < s$. By choice of W_j, W_j is the last write by writer j for which $Write(W_j) < s$. Since $Write(W_k) < Scan(W) < Scan(W_j)$, by Theorem 7, we have:

$$OVN[j,k]_s = VN[k,j]_s.$$

By choice of s,

$$OVN[j,k]_s = OVN[j,k]_R.$$

Let u be the state following $1Scan(R)_k$. By proposition 19.2, $Scan(W) < Start(R)$, implying $Scan(W) < Start(R) < u < 2Scan(R)_j < s$. Since, by Proposition 19.4, there are no writes W'_k by writer k for which $Scan(W) < Write(W'_k) < s$, $VN[k,j]_{s'}$ equals a constant for states s', $Scan(W) < s' < s$; in particular,

$$VN[k,j]_s = VN[k,j]_u.$$

By choice of u,

$$VN[k,j]_u = VN[k,j]_R.$$

Putting the above equations together, we get:

$$OVN[j,k]_R = OVN[j,k]_s = VN[k,j]_s = VN[k,j]_u = VN[k,j]_R.$$

Since $VN[k,j]_R = OVN[j,k]_R$, we have $k \in VNS(j)_R$. Since k was an arbitrary element of $VNS(i)_R$, $VNS(i)_R \subset VNS(j)_R$. Since $i \in VNS(j)_R$ but by Proposition 19.3, $i \notin VNS(i)_R$, $VNS(i)_R$ is a proper subset of $VNS(j)_R$.

(b) Case 2: $i < j$. The proof of this case is very similar to, although not identical to, that of the first case, so we will omit many of the details. Let W_j be the last write by writer j for which $Write(W_j) < 1Scan(R)_j$. Let s be the state following $1Scan(R)_j$. As before, we can show $Write(W) < Scan(W_j)$, and thus $Scan(W) < Scan(W_j)$.

Let k be any writer for which $k \in VNS(i)_R$, and let W_k be the last write by writer k for which $Write(W_k) < Scan(W)$. Then by Proposition 19.1, W_k is also the last write by writer k for which $Write(W_k) < 1Scan(R)_j$ since $1Scan(R)_j < 2Scan(R)_k$. As before, W_j and W_k are the last writes by writers j and k respectively for which $Write(W_j) < s$ and $Write(W_k) < s$. Again, we have $OVN[j,k]_s = VN[k,j]_s$. Again, $OVN[j,k]_s = OVN[j,k]_R$. Since there are no writes W'_k by writer k for which $Scan(W) < Write(W'_k) < 2Scan(R)_k$ and $Scan(W) < s < 2Scan(R)_k$, we have $VN[k,j]_s = VN[k,j]_u = VN[k,j]_R$ where u is the state following $2Scan(R)_k$. Thus $VN[k,j]_R = OVN[j,k]_R$ and as before, $VNS(i)_R$ is a proper subset of $VNS(j)_R$.

Since $i \in VNS(j)_R$ implies $i \neq j$, the proofs of the above two cases complete the proof of the proposition. □

Proposition 19.6 *Let j be any writer, $j \neq i$.*

(a) If $j < i$ and if there is some write W_j by writer j such that $2Scan(R)_j < Write(W_j) < 3Scan(R)_j$, then $OVN[j,i]_R = VN[i,j]_R$.

(b) If $i < j$ and if there is some write W_j by writer j such that $1Scan(R)_j < Write(W_j) < 2Scan(R)_j$, then $OVN[j,i]_R = VN[i,j]_R$.

Proof of Proposition 19.6:

(a) Let W_j be the last write by writer j such that $2Scan(R)_j < Write(W_j) < 3Scan(R)_j$. Let s and t be the states following $2Scan(R)_j$ and $3Scan(R)_j$ respectively. Now since the last three scans of R see the same values for the VN's, $VN[j,j]_s = VN[j,j]_t$. Thus by Lemma 3 there exists at least one write W_j' by writer j such that $s < Scan(W_j') < Write(W_j') < t$; since W_j is the last write by writer j for which $s < Write(W_j) < t$, we consequently have $s < Scan(W_j) < Write(W_j) < t$. Note then that we have the following order:

$$1Scan(R)_i < 2Scan(R)_j < s < Scan(W_j) < 3Scan(R)_j < t < 3Scan(R)_i.$$

By choice of t,

$$OVN[j,i]_R = OVN[j,i]_t.$$

Since $1Scan(R)_i < t < 3Scan(R)_i$, by Proposition 19.1 we have

$$VN[i,j]_R = VN[i,j]_t.$$

Also by Proposition 19.1, W is the last write by writer i for which $Write(W) < t$. Furthermore, by choice of W_j, W_j is the last write by writer j for which $Write(W_j) < t$. By Proposition 19.1, $Write(W) < 1Scan(R)_i$ thus $Write(W) < 1Scan(R)_i < Scan(W_j)$, and by Theorem 7 we have

$$VN[i,j]_t = OVN[j,i]_t.$$

Putting all these equations together yields:

$$VN[i,j]_R = VN[i,j]_t = OVN[j,i]_t = OVN[j,i]_R.$$

(b) Since $i < j$ implies $1Scan(R)_i < 1Scan(R)_j < 2Scan(R)_j < 3Scan(R)_i$, the second part of the proof of the proposition follows exactly like the first; $1Scan(R)_j$ replaces $2Scan(R)_j$, and $2Scan(R)_j$ replaces $3Scan(R)_j$.

This completes the proof of Proposition 19.6. □

Proposition 19.7 *Let j be any writer, $j \neq i$.*

(a) If $j < i$ and there is some write W_j by writer j such that $2Scan(R)_j < write(W_j) < 3Scan(R)_j$ then $|VNS(j)_R| > |VNS(i)_R|$.

(b) If $i < j$ and there is some write W_j by writer j such that $1Scan(R)_j < write(W_j) < 2Scan(R)_j$ then $|VNS(j)_R| > |VNS(i)_R|$.

Proof of Proposition 19.7: This follows directly from Proposition 19.5 and Proposition 19.6. □

Proposition 19.8 *Let j be any writer, $j \neq i$.*

(a) *If $j < i$ and there is some write W_j by writer j such that $2Scan(R)_j < write(W_j) < 3Scan(R)_j$ then $N(i)_R = 0$.*

(b) *If $i < j$ and there is some write W_j by writer j such that $1Scan(R)_j < write(W_j) < 2Scan(R)_j$ then $N(i)_R = 0$.*

Proof of Proposition 19.8:

(a) Let x and y be the states following $2Scan(R)_j$ and $3Scan(R)_j$ respectively. Then $VN[j,j]_x = VN[j,j]_y$. Thus by Lemma 3, we may let W_j and W_j' be the last two writes by writer j such that

$$x < Scan(W_j') < Write(W_j') < Scan(W_j) < Write(W_j) < y.$$

Let s, t, u, and v be the states following $Scan(W_j')$, $Write(W_j')$, $Scan(W_j)$, and $Write(W_j)$ respectiveley. Then by Proposition 19.1,

$$OVN[i,j]_s = OVN[i,j]_u = OVN[i,j]_R.$$

Also, by Lemma 1, we have

$$\begin{aligned} VN[j,i]_v &\neq OVN[i,j]_u \\ VN[j,i]_t &\neq OVN[i,j]_s \\ PVN[j,i]_v &= VN[j,i]_t. \end{aligned}$$

Since W_j is the last write by writer j for which $Write(W_j) < 3Scan(R)_j$, we have

$$\begin{aligned} VN[j,i]_R &= VN[j,i]_v \\ PVN[j,i]_R &= PVN[j,i]_v \end{aligned}$$

Putting this all together, we get:

$$VN[j,i]_R = VN[j,i]_v \neq OVN[i,j]_u = OVN[i,j]_R$$

$$PVN[j,i]_R = PVN[j,i]_v = VN[j,i]_t \neq OVN[i,j]_s = OVN[i,j]_R.$$

We conclude $N(i)_R = 0$.

(b) The second part of the proof of the proposition follows exactly like the first if we replace $2Scan(R)_j$ by $1Scan(R)_j$ and replace $3Scan(R)_j$ by $2Scan(R)_j$.

This completes the proof of Proposition 19.8. □

Proposition 19.9 *Let j be any writer, $j \neq i$.*

(a) *If $j < i$ then there is no write by writer j such that $2Scan(R)_j < Write(W_j) < 3Scan(R)_j$.*

(b) If $i < j$ then there is no write by writer j such that $1Scan(R)_j < Write(W_j) < 2Scan(R)_j$.

Proof of Proposition 19.9: Assume otherwise. Then by Proposition 19.7 and Proposition 19.8, we have:

$$|VNS(i)_R| + N(i)_R = |VNS(i)_R| < |VNS(j)_R| \leq |VNS(j)_R| + N(j)_R.$$

This contradicts the fact that $F(R) = i$ and the proposition is thus proved by contradiction. □

Proposition 19.10 *Let j be any writer, $j \neq i$.*

(a) If $j < i$ then for all states u, $2Scan(R)_j < u < 3Scan(R)_j$, and all writers k,

$$\begin{aligned} VN[j,k]_u &= VN[j,k]_R \\ OVN[j,k]_u &= OVN[j,k]_R \\ PVN[j,k]_u &= PVN[j,k]_R. \end{aligned}$$

(b) If $i < j$ then for all states u, $1Scan(R)_j < u < 2Scan(R)_j$, and all writers k,

$$\begin{aligned} VN[j,k]_u &= VN[j,k]_R \\ OVN[j,k]_u &= OVN[j,k]_R \\ PVN[j,k]_u &= PVN[j,k]_R. \end{aligned}$$

Proof of Proposition 19.10: This proposition is a direct consequence of Proposition 19.9. □

We now use these propositions to complete the proof of Theorem 19. Let s be the state following $2Scan(R)_i$. Note that for all writers j, if $j < i$ then we have $2Scan(R)_j < s < 3Scan(R)_j$, and if $i < j$ then we have $1Scan(R)_j < s < 2Scan(R)_j$. Then by Proposition 19.10, we have

$$VN[j,k]_R = VN[j,k]_s$$

$$OVN[j,k]_R = OVN[j,k]_s$$

$$PVN[j,k]_R = PVN[j,k]_s$$

for all writers j and k. But this means that $F(s) = F(R) = i$.

Let W_i be the last potent write for which $Write(W_i) < s$. Since F remains constant between consecutive $Write$ actions of potent writes, if t is the state following $Write(W_i)$ then $F(t) = F(s) = i$. Since W_i is potent, this implies W_i was written by writer i. Since $F(s') = i$ for all states s', $t \leq s' \leq s$, by definition of impotent writes there can be no impotent write W_i' by writer i for which $t < Write(W_i') < s$. Then since W_i is the last potent write by writer i for

which $Write(W_i) < s$, W_i is the last write, potent or impotent, by writer i for which $Write(W_i) < s$. By Proposition 19.1, W is the last write by writer i for which $Write(W) < s$. Therefore $W = W_i$.

Since W is thus potent, $Atomic(W) = Write(W)$. Since W is the last potent write for which $Write(W) < s$, there can be no other writes W' such that $Atomic(W) < Atomic(W') < s$ as there are no potent writes W'' in this interval before which such $Atomic(W')$ could be inserted. This contradicts our initial assumption, upon which this whole sequence of propositions was based, that such a W' exists. Thus our initial assumption is incorrect; there exists no write W' such that $Atomic(W) < Atomic(W') < Start(R)$.

This (finally) completes proof of Theorem 19. □

We will now use Theorem 19 to place the $Atomic(R)$ actions for reads R. Let R be any read. Then $Atomic(R)$ will be placed as follows:

1. If R did not time out, then let $i = F(R)$, and let W be the last write by writer i for which $Write(W) < 3Scan(R)_i$ as we did in the proof of Theorem 19. Then we have two cases:

   (a) If $Start(R) < Atomic(W)$ then by Theorem 19, $Start(R) < Atomic(W) < Finish(R)$. Thus if we insert $Atomic(R)$ immediately following $Atomic(W)$ it is clear that $Start(R) < Atomic(R) < Finish(R)$. Also, since Theorem 19 states $Value(R) = Value(W)$, it is clear that R returns the value of the last write W for which $Atomic(W) < Atomic(R)$.

   (b) If $Atomic(W) < Start(R)$ then we will insert $Atomic(R)$ immediately following $Start(R)$. It is clear that $Start(R) < Atomic(R) < Finish(R)$. Also, since Theorem 19 states $Value(R) = Value(W)$ and that there are no writes W' for which $Atomic(W) < Atomic(W') < Start(R)$, it is clear that R returns the value of the last write W for which $Atomic(W) < Atomic(R)$.

2. If R did time out, then we know from the fact that it timed out that, for some writer i, R saw the contents of writer i's register change twice. Since the values in writer i's register that are visible to readers (the $VN[i,j]$, $OVN[i,j]$, $PVN[i,j]$, and $Value[i]$) change only at the points $Write(W')$ for writes W' by writer i that do not time out, the two observed changes must have been caused by separate writes by writer i. The write that caused the second of these observed changes, call it W', must have begun after the first finished. Thus we have $Start(R) < Scan(W') < Write(W') < Finish(R)$. Whether W' is potent or impotent, we have $Scan(W') < Atomic(W') \leq Write(W')$, thus if we insert $Atomic(R)$ immediately following $Write(W')$ it is clear that we will have $Start(R) < Atomic(R) < Finish(R)$. Also, since the algorithm returns $Value[i]$, it is clear that $Value(R) = Value(W')$. Thus R returns the value written by the last write W' for which $Atomic(W') < Atomic(R)$.

Here, as was the case when we placed the *Atomic* actions for impotent writes and writes that timed out, we may have to insert several *Atomic* read actions following a given *Atomic* write action; again, this causes no problem.

15 Conclusion

Thus for every read R and every write W we have placed internal actions $Atomic(R)$ and $Atomic(W)$ such that:

1. $Start(W) < Atomic(W) < Finish(W)$.
2. $Start(R) < Atomic(R) < Finish(R)$.
3. If W_R is the last write for which $Atomic(W_R) < Atomic(R)$ then $Value(R) = Value(W_R)$.

This completes the proof of correctness.

Having thus completed our proof of correctness it is appropriate to reflect on the purpose of this paper, to provide intuitive explanation and rigorous proof of the correctness of the multi-writer, multi-reader atomic register algorithm presented in [PB]. We have gone about this in several ways. First, the algorithm is presented, at an intuitive level, before the proof of correctness. This should hopefully arm readers of the proof with an understanding of what needs to be proved and why. Second, the approach to the problem is that taken in [BB]. An attempt is made to understand what different reads and writes do so that their *Atomic* actions may be placed in an appropriate and intuitively reasonable manner. Third, the proof has examined the algorithm at a finer level of detail than that presented in [PB]. Arguments are presented at the level of the individual reads of writers' registers and not at the level of scans as a whole. The result of this detailed proof was to find two problems with the algorithm. The detailed approach to proof is not, however, without its faults; it is possible to be so attentive to detail that the proof becomes little more than an exercise in symbol manipulation to those not already intimiately familiar with the algorithm. Thus while care was taken to present detail where necessary, as was the case with arguments about individual reads in scans, some arguments, particularly those dealing with the choice of VN's and PVN's by successive writes, have been presented in somewhat less detail. It is hoped then that one will find in this paper a clear survey of the algorithm in question in addition to a rigorous, but not overburdened, proof of correctness.

16 References

[BB] Bloom, Bard, "Constructing Two-Writer Atomic Registers," Proceedings of the Symposium on Principles of Distributed Computing, pp. 249-259, August 1987.

[IL] Israeli, A. and Ming Li, manuscript.

[LL] Lamport, Leslie, "On Interprocess Communication," Digital Systems Research Center Report 8.

[LT] Lynch, Nancy A. and Mark R. Tuttle, "Hierarchical Correctness Proofs for Distributed Algorithms," Proceedings of the Symposium on Principles of Distributed Computing, pp. 137-151, August 1987.

[LV] Li, Ming, and Paul Vitanyi, manuscript.

[PB] Peterson, Gary L. and James E. Burns, "Concurrent Reading While Writing II: The Multi-writer Case," Proceedings of the Symposium on Foundations of Computer Science, pp. 383-392, October 1987.