THE ELUSIVE ATOMIC REGISTER REVISITED

Ambuj K. Singh*, James H. Anderson,
and Mohamed G. Gouda

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-86-30 December 1986

Abstract

A new construction of a 1-writer/m-reader/n-bit atomic register using $O(m^2n)$ 1-writer/1-reader/1-bit atomic registers is presented. This construction is more efficient, i.e, uses less registers, than previous constructions.

*Work supported by NSF Grant ECS 83-04734 and Office of Naval Research Contract N00014-86-K-0182

1 Introduction

The currently accepted theory of concurrent computing is deeply rooted in the concept of atomic registers. An *atomic register* is one that is read or written by one or more processes according to the following assumption. If some reading or writing operations of the register are enabled simultaneously in different processes, then these operations are executed in sequence, one after the other, and not concurrently. This assumption strongly suggests the well-known interleaving semantics of concurrent computations. Therefore, the validity of this assumption is a corner stone in establishing the validity of the present theory of concurrent computing.

One way to check the validity of this assumption is to start with a more realistic model of a register, in particular one that admits concurrent reading and writing operations by different processes; and to then show that an atomic register can be constructed using a number of these registers. Informally, the construction of an atomic register consists of a set of registers, along with some programs that access them. Any process that needs to read or write the constructed atomic register invokes one of these programs. Different programs can be invoked by different processes concurrently; the net effect, however, should resemble that of a serial invocation. The programs are restricted to having no wait statements, and no unbounded loops. The first restriction guarantees that a process trying to read or write the constructed register should be able to do so in a finite time, regardless of the activities of other processes. The second restriction guarantees that no busy wait statements can be introduced into the construction programs, for the same reason.

Peterson [Pe 83] was the first to suggest the problem of constructing atomic registers from safe registers. A *safe register* is one that can be read and written concurrently by different processes: a read operation that overlaps a write operation may return any value from the value domain of the register. The leap from safe registers to atomic registers is quite large; fortunately, it can be divided into a number of steps. In order to identify these steps, the following notation is used in defining registers. A register, safe or atomic, is defined by the triple k/m/n iff it can be written by k processes, read by m processes, and can store an n-bit value. When the number of bits is arbitrary, the third value can be omitted; then, the register

is defined by the pair k/m. Based on this notation, the most primitive register is 1/1/1 safe, and the most sophisticated is k/m/n atomic. Figure 1 depicts two chains of register constructions that lead from 1/1/1 safe to k/m/n atomic registers. (Each construction is labeled by a reference to the paper in which it is presented.)

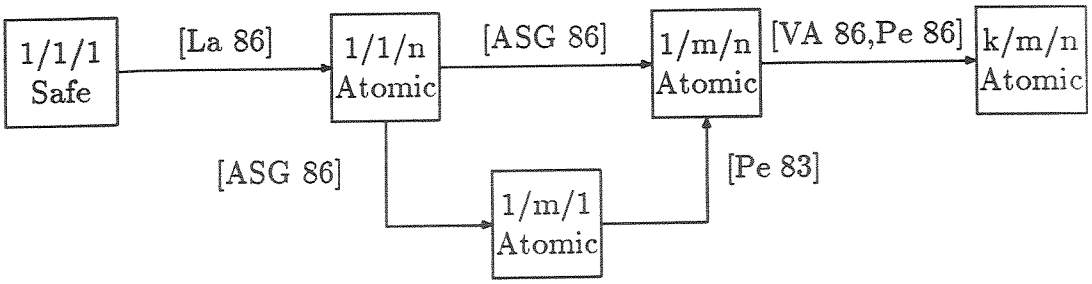

Figure 1: Two Chains of Register Constructions.

The problem of constructing 1/m atomic register from 1/1 atomic registers has been mentioned as an open problem by Lamport [La 86] and Vitanyi and Awerbuch [VA 86]. A solution to the problem was presented in [ASG 86]. This solution, though easy to explain and understand, has an exponential complexity. In this paper, a polynomial solution to the problem is presented.

The rest of this paper is organized as follows. In Section 2, the problem of constructing 1/m atomic registers from 1/1 atomic registers is formally defined. In Section 3, the construction of a 1/m atomic register from 1/1 atomic registers is presented. In Section 4, this construction is proved to be correct. The time and space complexity of the construction is discussed in Section 5. Section 6 contains concluding remarks.

2 Register Construction

Definition 1: A *construction* of a 1/m register using 1/1 atomic registers consists of

- a set of 1/1 atomic registers called *internal* registers,

- (m+1) 1/1 atomic registers collectively called *interface* registers and individually named $input, output_0, \ldots, output_{m-1}$, and
- (m+1) programs named $writer$, $reader_0, \ldots, reader_{m-1}$,

such that the following three conditions hold.

i. Each internal register has an initial value. Any internal register is written by exactly one program, and is read by exactly one program.

ii. Each interface register has an initial value. Interface register *input* is read only by program *writer* and written by no program in the construction. Each interface register $output_i$ is written only by program $reader_i$ and read by no program in the construction.

iii. Each program has the following structure:

forever do wait; <body> **od**

such that the following three conditions hold.

a. The program can wait at the **wait** statement for an arbitrary length of time, possibly indefinitely.

b. <body> consists of one or more statements, each of which can access at most one register. <body> has no **wait** statements and no unbounded loops.

c. Program *writer* reads register *input* exactly once in its body; program $reader_i$ writes register $output_i$ exactly once in its body. □

The following definitions apply to an arbitrary register construction.

Definition 2: Let P be any program in the construction. An *event* of P is a triple, denoted $P:i:j$, where i is a natural number and j is a statement in P. □

Definition 3: A *state* of the construction is defined by a value for each register and an event for each program in the construction. The *initial state* of the construction is defined by the initial value for each register and the

event $P:0$:**wait** for each program P in the construction. □

Definition 4: Let s and s' be two states of the construction, and let $e = P:i:j$ be the event of program P in state s. State s' is said to *follow* state s *over* e, denoted $s \xrightarrow{e} s'$, iff the following two conditions hold.

i. The register values in state s' are the result of executing statement j of program P in state s.

ii. The event for each program other than P in state s' is the same as the event for that program in state s. The event for program P in state s' is $P:i':j'$ where:

> if j is the last statement in the body of P then j' is the **wait** statement and $i' = i + 1$, else j' is the statement following statement j in P and $i' = i$. □

Definition 5: A *history* of the construction is a sequence $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} s_2 \ldots$ where s_0 is the initial state of the construction and for each i, state s_{i+1} follows state s_i over event e_i. □

Definition 6: Let $stmt_0, stmt_1, .., stmt_{k-1}$ denote the sequence of statements in the body of program P. Then, for any natural number i, the sequence of events $P:i:stmt_0$, $P:i:stmt_1$, .., $P:i:stmt_{k-1}$ is called an *operation* of P, and is denoted $P:i$. (Informally, an operation $P:i$ is the sequence of events corresponding to the ith execution of program P in some history.) If P is the writer program then $P:i$ is called a *write operation*; Otherwise, P is a reader program and $P:i$ is called a *read operation*. □

Definition 7: An operation $P_1:i_1$ precedes another operation $P_2:i_2$ in a history h if the last event of the operation $P_1:i_1$ precedes the first event of the operation $P_2:i_2$ in h. □

Notice that from definitions 4 and 7, operation $P:i$ precedes operation $P:i+1$, for any program P and any i.

Definition 8: A history h is called *proper* iff the initial write operation, $W:0$, precedes all read operations in h.

Definition 9: Let w be any write operation in a history h. By definition, w contains exactly one event in which the value of the interface register *input* is read; let this value be v. In this case, w is said to *assign* v to the 1/m register. Similarly, let r be any read operation in a history h. By definition, r contains exactly one event in which a value is written to some interface register $output_i$; let this value be v. In this case, r is said to *return* v from the 1/m register. □

Definition 10: Let h be any proper history of the construction. h is said to be atomic iff there exists a function $\phi : \{all\ read\ operations\ in\ h\} \rightarrow \{0, 1, 2 \ldots\}$ such that the following three conditions hold.

i. *Integrity:* For each read operation r in h, if r returns a value from the 1/m register then the write operation $W : \phi(r)$ assigned that value to the 1/m register.

ii. *Safety:* For each read operation r in h, r does not precede the write operation $W : \phi(r)$ and the write operation $W : \phi(r) + 1$ does not precede r. This condition ensures that a read operation does not return a value from the future or the far past.

iii. *Precedence:* For any two read operations r, r' in h, if r precedes r' then $\phi(r) \leq \phi(r')$. □

Definition 11: A 1/m register construction is *atomic* iff all its proper histories are atomic. □

3 Construction of a 1/m atomic register

3.1 Architecture

Our construction of 1/m atomic register has the following internal registers.

- A register named WW is written and read by the *writer*. This register stores the value written to the 1/m register by the most recent write operation.

- A register named WR_i is written by the *writer* and read by $reader_i$, $i = 0, .., m-1$. The fields of each WR_i are $old, new, seq_0, .., seq_{m-1}$, and *done*. Informally, *old* is the value of the 1/m register before the most recent write operation; *new* is the value written to the 1/m register by the most recent write operation; seq_i is the ith sequence number written by the most recent write operation; *done* is a boolean indicating if the most recent write operation is completed.

- A register named R_iW is written by $reader_i$ and read by the *writer*, $i = 0, .., m-1$. Each register R_iW stores the ith sequence number read by the most recent read operation of $reader_i$.

- A register named R_iR_j is written by $reader_i$ and read by $reader_j$, where $i < j$. The fields of each register R_iR_j are old, new, seq_i, and $flag$. Informally, *old* is the "old" value of the 1/m register as read by the most recent read operation of $reader_i$; *new* is the "new" value of the 1/m register as read by the most recent read operation of $reader_i$; seq_i is the ith sequence number read by the most recent read operation of $reader_i$; $flag$ is a boolean indicating if the most recent read operation of $reader_i$ returned the "new" value of the 1/m register.

3.2 Initial State

The initial register values are as follows:

$$\begin{array}{ll} EW = (y) & R_iE = (x) \\ WW = (x) & WR_i = (y, x, 0, .., 0, true) \\ R_iW = (0) & R_iR_j = (y, x, 0, true) \end{array}$$

where x and y are arbitrary but distinct values of the constructed register.

3.3 Algorithm

The body of writer W is as follows:

read new **from** EW;
read old **from** WW;
if $old \neq new$ **then**
  for $k = 0$ **to** $m-1$ **do read** seq_k **from** R_kW; $seq_k := (seq_k + 1)\ mod\ 3$ **od**;
  for $k = m-1$ **to** 0 **do write** $(old,\ new,\ seq_0, .., seq_{m-1},\ false)$ **to** WR_k **od**;
  for $k = 0$ **to** $m-1$ **do write** $(old,\ new,\ seq_0, .., seq_{m-1},\ true)$ **to** WR_k **od**;
  write new **to** WW;
fi

A write operation assigns a new value to the 1/m register only if the new value is different from the current value in the register. If so, it computes a sequence number for each reader; the sequence number for any reader is the successor of the sequence number read by the last read operation of that reader. It then writes to the readers in two sweeps with *done* set to $false$ in the first sweep and set to $true$ in the second.

The body of $reader_i$ is as follows:

read $(old,\ new,\ seq_0, .., seq_{m-1},\ done)$ **from** WR_i;
write seq_i **to** R_iW;
for $k = 0$ **to** $i-1$ **do read** $(old_k,\ new_k,\ seq''_k,\ flag_k)$ **from** R_kR_i **od**;
read $(old',\ new',\ seq'_0, .., seq'_{m-1},\ done')$ **from** WR_i;

$p_i := (old = old') \wedge (new = new') \wedge (\forall j : 0 \leq j < m : seq_j = seq'_j)$;
for $k = 0$ **to** $i-1$ **do** $p_k := p_i \wedge flag_k \wedge (old = old_k) \wedge (new = new_k) \wedge (seq_k = seq''_k)$ **od**;
$flag := p_0 \vee p_1 \vee .. \vee p_{i-1} \vee (p_i \wedge done \wedge done')$;

for $k = i+1$ **to** $m-1$ **do write** $(old',\ new',\ seq'_i,\ flag)$ **to** R_iR_k **od**;
if $flag$ **then** $retval := new'$
  elsif p_i **then** $retval := old'$ **else** $retval := new$ **fi**
write $retval$ **to** R_iE;

A read operation of R_i computes the predicates $p_0, .., p_i$. It will find a predicate $p_j, 0 \leq j < i$, $true$ if a preceding read operation of some R_j has returned the "new" value of the 1/m register; in that case it will also return the "new" value. If it does not overlap the most recent write operation, then it will find predicate p_i and booleans $done, done'$ to be $true$ and will also the "new" value of the 1/m register.

4 Proof of Correctness

The above construction is proved correct by defining a function ϕ for each proper history of the construction, then showing that the defined ϕ meets the three conditions of *integrity*, *safety*, and *precedence* in Definition 10. But first, some notation is introduced.

Let f be a field of a register R in the construction. Let s be any state in some proper history of the construction. Then, $(R.f)_s$ denotes the value of the field f of register R in state s. For example, if s is the initial state of the construction then $(WR_0.old)_s = y$.

Let $P{:}i$ be any operation in a proper history h and let var be a local variable in P. Then, $(P{:}i).var$ denotes the value of var in the state immediately following the last event of operation $P{:}i$ in h.

The following definition of ϕ is for any proper history h of the construction. Let $R_i{:}k$ be any read operation in h. Assume that it reads its first value of register WR_i from the write operation $W{:}u$ and its second value of register WR_i from the write operation $W{:}v$. (This implies that $u \leq v$.) Then define $\phi(R_i{:}k)$ as follows.

$$\phi(R_i{:}k) = \begin{cases} v & \text{if } (R_i{:}k).flag \\ v-1 & \text{if } \neg(R_i{:}k).flag \wedge (R_i{:}k).p_i \\ u & \text{if } \neg(R_i{:}k).flag \wedge \neg(R_i{:}k).p_i \end{cases}$$

This ϕ satisfies the *integrity* condition because if $R_i{:}k$ returns a value from the 1/m register then the write operation $W{:}\phi(r)$ indeed assigned that value to the register.

To show that ϕ satisfies the *safety* condition, consider the following. If $\phi(R_i{:}k) = u$ or v then the read operation $R_i{:}k$ returns a value assigned by the write operation $W{:}\phi(R_i{:}k)$; therefore, $R_i{:}k$ does not precede $W{:}\phi(R_i{:}k)$ and the write operation $W{:}\phi(R_i{:}k)+1$ does not precede the read operation $R_i{:}k$. Now, consider the remaining possibility, i.e. $\phi(R_i{:}k) = v-1$. If $u < v$ then, clearly, the read operation $R_i{:}k$ overlapped the write operation $W{:}v$. If $u = v$ then since, $\neg(R_i{:}k).done \vee \neg(R_i{:}k).done'$, the read operation $R_i{:}k$ overlapped the write operation $W{:}v$. Thus, $R_i{:}k$ does not precede the write operation $W{:}v-1$ and the write operation $W{:}v$ does not precede the read operation $R_i{:}k$. Therefore, ϕ satisfies the *safety* condition.

Next, it is shown that ϕ satisfies the *precedence* condition. Let r and r' be any two read operations in h such that r precedes r'; then, it is required to show that $\phi(r) \leq \phi(r')$. There are three cases to consider for this proof:

Case 1: $r = R_i{:}k$ and $r' = R_i{:}k{+}1$.

Case 2: $r = R_i{:}k$ and $r' = R_j{:}n$, where $i < j$.

Case 3: $r = R_i{:}k$ and $r' = R_j{:}n$, where $i > j$.

Proof of Case 1: Assume that read operation $R_i{:}k$ reads its first value of register WR_i from the write operation $W{:}u$ and its second value of register WR_i from the write operation $W{:}u'$. Assume that read operation $R_i{:}k{+}1$ reads its first value of register WR_i from the write operation $W{:}v$ and its second value of register WR_i from the write operation $W{:}v'$. Then, $u \leq u' \leq v \leq v'$, $\phi(R_i{:}k) \leq u'$, and $\phi(R_i{:}k{+}1) \geq min(v, v' - 1)$.

If $u' < v$ then $\phi(R_i{:}k) \leq v - 1 \leq \phi(R_i{:}k{+}1)$. If $v < v'$ then $\phi(R_i{:}k) \leq v \leq \phi(R_i{:}k{+}1)$. So, assume that $u' = v = v'$ in the rest of the proof.

If $(R_i{:}k).flag$ is *true* then, by Lemma 1 in the Appendix, $(R_i{:}k{+}1).flag$ will also be *true*; therefore, $\phi(R_i{:}k) \leq \phi(R_i{:}k{+}1)$. If $(R_i{:}k).flag$ is *false* then either $u < u'$ or $u = u'$. If $u < u'$ then, since $(R_i{:}k).flag$ is false, $\phi(R_i{:}k) \leq u' - 1$. If $u = u'$ then $(R_i{:}k).p_i$ will be true; therefore, $\phi(R_i{:}k) = u' - 1$. Thus, if $(R_i{:}k).flag$ is false then $\phi(R_i{:}k) \leq u' - 1 = v - 1 \leq \phi(R_i{:}k{+}1)$. This completes the proof. □

Proof of Case 2: Assume that read operation $R_i{:}k$ reads its first value of register WR_i from the write operation $W{:}u$ and its second value of register WR_i from the write operation $W{:}u'$. Assume that read operation $R_j{:}n$ reads its first value of register WR_i from the write operation $W{:}v$ and its second value of register WR_i from the write operation $W{:}v'$. Then, $u \leq u' \leq v \leq v'$, $\phi(R_i{:}k) \leq u'$, and $\phi(R_j{:}n) \geq min(v, v' - 1)$.

If $u' < v$ then $\phi(R_i{:}k) \leq v - 1 \leq \phi(R_j{:}n)$. If $v < v'$ then $\phi(R_i{:}k) \leq v \leq \phi(R_j{:}n)$. So, assume that $u' = v = v'$ in the rest of the proof.

If $(R_i{:}k).flag$ is *true* then, by Lemma 2 in the Appendix, $(R_j{:}n).flag$ will also be *true*; therefore, $\phi(R_i{:}k) \leq \phi(R_j{:}n)$. If $(R_i{:}k).flag$ is *false* then, by reasoning as in Case 1, $\phi(R_i{:}k) \leq u' - 1 = v - 1 \leq \phi(R_j{:}n)$. This completes the proof. □

Proof of Case 3: Assume that read operation $R_i:k$ reads its first value of register WR_i from the write operation $W:u$ and its second value of register WR_i from the write operation $W:u'$. Assume that read operation $R_j:n$ reads its first value of register WR_i from the write operation $W:v$ and its second value of register WR_i from the write operation $W:v'$. Then, $u \leq u' \leq v+1 \leq v'+1$, $\phi(R_i:k) \leq u'$ and $\phi(R_j:n) \geq min(v, v'-1)$.

If $u' < v$ then $\phi(R_i:k) \leq v-1 \leq \phi(R_j:n)$. The remaining possibilities, $u' = v$ or $u' = v+1$, are considered next.

Consider the first possibility, i.e., $u' = v$. If $v < v'$ then $\phi(R_i:k) \leq v \leq \phi(R_j:n)$. So, assume $v = v'$. If $(R_i:k).flag$ is $false$ then, by reasoning as in Case 1, $\phi(R_i:k) \leq u'-1 \leq \phi(R_j:n)$. If $(R_i:k).flag$ is $true$ then, by Lemma 3 in the Appendix, $(R_j:n).flag$ will also be true; therefore, $\phi(R_i:k) \leq \phi(R_j:n)$.

Consider the second possibility, i.e., $u' = v+1$. By Lemma 4 in the Appendix, $(R_i:k).flag$ is $false$; therefore, by reasoning as in Case 1, $\phi(R_i:k) \leq v$. If $v < v'$ then $\phi(R_i:k) \leq v \leq \phi(R_j:n)$. If $v = v'$ then $(R_j:n).p_j$, $(R_j:n).done$, and $(R_j:n).done'$ will all be $true$; therefore, $\phi(R_j:n) = v \geq \phi(R_i:k)$. This completes the proof. □

5 Construction Complexity

We measure the space complexity of a 1/m/n atomic register construction by the number of 1/1 atomic bits used to construct the internal registers. The number of 1/1 atomic bits used in our construction is summarized below.

- Register WW uses n 1/1 atomic bits.
- Register WR_i, $0 \leq i < m$, uses $2m + 2n + 1$ 1/1 atomic bits.
- Register R_iW, $0 \leq i < m$, uses 2 1/1 atomic bits.
- Register R_iR_j, $0 \leq i < j < m$, uses $2n + 3$ atomic bits.

Thus, the space complexity of our construction is $O(m^2n)$.

Any 1/m/n atomic register construction must have a register between any two readers in order to satisfy the *precedence* condition. Also, the size

of any such register should be $O(n)$ in order to distinguish between different write operations. This means that the lower bound on the space complexity of 1/m/n atomic register construction is $O(m^2n)$ which is the complexity of our construction.

The time complexity of a construction is measured by the number of statements executed by each program in the construction. In our construction, each program executes $O(m)$ statements.

In any 1/m/n atomic register construction, the writer must write to every reader; therefore, the lower bound on the time complexity of the writer program is $O(m)$. Also, any reader must read to, or write from, every other reader; therefore, the lower bound on the time complexity of a reader program is also $O(m)$. The programs in our construction meet these lower bounds.

6 Discussion

In our construction, each program has about 10 statements, yet, its proof of correctness occupies about 10 pages. This is because a construction has to be proved atomic for all histories any of which may contain an arbitrary interleaving of read and write operations. In fact, the correctness proof would have been even more complex if we focussed on the actual times at which events occurred. Using the function ϕ, which refers only to the relative ordering of read and write operations, made the proof manageable.

Our definition of atomicity is equivalent to that given by Misra in [Mi 86]. His axioms for atomicity in essence require that all read and write operations be shrunk to a point; such a shrinking of operations is possible iff a function ϕ that meets the three conditions of our definition exists.

Atomicity is similar to the condition of serializability as defined in database literature; the only difference between the two is one of parlance. A transaction is usually assumed to consist of a number of operations; the equivalent serial schedule for serializability preserves the relative ordering of transactions in the same way the equivalent serial schedule for atomicity preserves the relative ordering of operations. As observed by Lamport in [La 86] the distinction between serializability and atomicity vanishes if we view a transaction as a 'higher-level' operation on the 'atomic database.'

After developing the constructions in this paper, it came to our attention that Kirousis, Kranakis, and Vitanyi [KKV 86] have developed another construction of a 1/m/n atomic register. There construction, however, is completely different from ours; in fact its space complexity is $O(m^3 + m^2 n)$ which is greater than the space complexity of our construction.

Appendix: Lemmas and Propositions

The proof of correctness of the construction is based on 4 lemmas; these are stated and proved in this appendix. The proof of these lemmas is, in turn, based on 5 propositions; these propositions are also stated and proved here. The following discussion and proofs are assumed to be in the context of a proper history h.

The following notation is adopted in the proofs that follow:

$Q(k,i,j,s)$ denotes the predicate $(R_iR_j.flag)_s \wedge (W:k).old = (R_iR_j.old)_s \wedge (W:k).new = (R_iR_j.new)_s \wedge (W:k).seq_i = (R_iR_j.seq)_s$.

$[s_0, s_1]$ denotes an interval of successive states in h that starts with state s_0 and ends with state s_1.

$[s_0, s_1)$ denotes an interval of successive states in h that starts with state s_0 and ends with the predecessor of state s_1.

$s_0 \prec s_1$ denotes that state s_0 precedes state s_1 in h.

$s_0 \preceq s_1$ denotes that state s_0 precedes or is the same as state s_1 in h.

$\oplus$ denotes modulo 3 addition.

Lemma 1 *Let $R_i:k$ and $R_i:k+1$ be two read operations in proper history h. Assume that read operation $R_i:k$ reads its second value of register WR_i and read operation $R_i:k+1$ reads both its values of register WR_i from the same write operation. If read operation $R_i:k$ evaluates its $flag$ to true then read operation $R_i:k+1$ also evaluates its $flag$ to true.*

Proof: Since $(R_i{:}k).flag$ is *true*, $(R_i{:}k).done' \vee \exists u < i :: (R_i{:}k).p_u$. If $(R_i{:}k).done'$ then $(R_i{:}k{+}1).done \wedge (R_i{:}k{+}1).done'$ and $R_i{:}k{+}1$ will evaluate its $flag$ to *true*. So, consider the other case i.e., $(R_i{:}k).p_u \wedge u < i$. Let $W{:}v$ be the write operation referred to in the lemma.

Let s_0 be the state immediately following the event in which write operation $W{:}v$ reads sequence number from reader R_u.

Let s_1 be the state immediately following the event in which write operation $W{:}v$ writes to reader R_0 for the second time.

Let s_2 be the state immediately following the event in which write operation $W{:}v{+}1$ writes to reader R_u for the first time.

Let s_3 be the state immediately following the event in which read operation $R_i{:}k$ writes sequence number to writer.

Let s_4 be the state immediately following the event in which read operation $R_i{:}k$ reads from reader R_u.

Let s_5 be the state immediately following the event in which read operation $R_i{:}k{+}1$ reads from reader R_u.

By the programs of writer and reader, $s_0 \prec s_1 \prec s_2$ and $s_3 \prec s_4 \prec s_5$. By assumption, $s_5 \prec s_2$. Since read operation $R_i{:}k$ evaluates its $flag$ to *true*, $Q(v, u, i, s_4)$ is *true*.

By Proposition 2, $s_0 \prec s_3$; therefore, $s_0 \prec s_4$. By Proposition 3, s_4 does not occur in the interval $[s_0, s_1]$; therefore, $s_1 \prec s_4$. Thus, $s_1 \prec s_4 \prec s_5 \prec s_2$ and $Q(v, u, i, s_4)$ is *true*; therefore, by Proposition 4, $Q(v, u, i, s_5)$ is *true*. Therefore, read operation $R_i{:}k{+}1$ will evaluate its $flag$ to *true*. □

Lemma 2 *Let $R_i{:}k$ and $R_j{:}n$ be two read operations in proper history h such that read operation $R_i{:}k$ precedes read operation $R_j{:}n$ and $i < j$. Assume that $R_i{:}k$ reads its second value of register WR_i and read operation $R_j{:}n$ reads both its values of register WR_j from the same write operation. If read operation $R_i{:}k$ evaluates its $flag$ to true then read operation $R_j{:}n$ also evaluates its $flag$ to true.*

Proof: Let $W{:}v$ be the write operation referred to in the lemma.

Let s_0 be the state immediately following the event in which write operation $W{:}v$ reads sequence number from reader R_i.

Let s_1 be the state immediately following the event in which write operation $W:v$ writes to reader R_0 for the second time.

Let s_2 be the state immediately following the event in which write operation $W:v+1$ writes to reader R_i for the first time.

Let s_3 be the state immediately following the event in which read operation $R_i:k$ writes sequence number to writer.

Let s_4 be the state immediately following the event in which read operation $R_i:k$ writes to reader R_j.

Let s_5 be the state immediately following the event in which read operation $R_j:n$ reads from reader R_i.

By the programs of writer and reader, $s_0 \prec s_1 \prec s_2$ and $s_3 \prec s_4 \prec s_5$. By assumption, $s_5 \prec s_2$. Since read operation $R_i:k$ evaluates its $flag$ to *true*, $Q(v,i,j,s_4)$ is *true*.

By Proposition 2, $s_0 \prec s_3$; therefore, $s_0 \prec s_4$. By Proposition 3, s_4 does not occur in the interval $[s_0, s_1]$; therefore, $s_1 \prec s_4$. Thus, $s_1 \prec s_4 \prec s_5 \prec s_2$ and $Q(v,i,j,s_4)$ is *true*; therefore, by Proposition 4, $Q(v,i,j,s_5)$ is *true*. Therefore, read operation $R_j:n$ will evaluate its $flag$ to *true*. □

Lemma 3 *Let $R_i:k$ and $R_j:n$ be two read operations in proper history h such that $R_i:k$ precedes $R_j:n$ and $j < i$. Assume read operation $R_i:k$ reads its second value of register WR_i and read operation $R_j:n$ reads both its values of register WR_j from the same write operation. If read operation $R_i:k$ evaluates its $flag$ to true then read operation $R_j:n$ also evaluates its $flag$ to true.*

Proof: Since $(R_i:k).flag$ is *true*, $(R_i:k).done' \vee \exists u < i :: (R_i:k).p_u$. If $(R_i:k).done'$ then, by the writer program and the assumption that $j < i$, $(R_j:n).done \wedge (R_j:n).done'$. Therefore, $R_j:n$ will evaluate its $flag$ to *true*. So, consider the other case i.e., $u < i \wedge (R_i:k).p_u$. Let $W:v$ be the write operation referred to in the lemma.

Let s_0 be the state immediately following the event in which write operation $W:v$ reads sequence number from reader R_u.

Let s_1 be the state immediately following the event in which write operation $W:v$ writes to reader R_0 for the second time.

Let s_2 be the state immediately following the event in which write operation

$W:v+1$ writes to reader R_u for the first time.

Let s_3 be the state immediately following the event in which read operation $R_i:k$ writes sequence number to writer.

Let s_4 be the state immediately following the event in which read operation $R_i:k$ reads from reader R_u.

By the programs of writer and reader, $s_0 \prec s_1 \prec s_2$ and $s_3 \prec s_4$. By assumption, $s_4 \prec s_2$. Since read operation $R_i:k$ evaluates its $flag$ to $true$, $Q(v,u,i,s_4)$ is $true$.

By Proposition 2, $s_0 \prec s_3$; therefore, $s_0 \prec s_4$. By Proposition 3, s_4 does not occur in the interval $[s_0, s_1]$; therefore, $s_1 \prec s_4$. Thus, $s_1 \prec s_4 \prec s_2$ and $Q(v,u,i,s_4)$ is $true$; therefore, by Proposition 5, $(WR_j.done)_{s_4} \vee \exists t < j :: Q(v,t,j,s_4)$ is $true$.

If $(WR_j.done)_{s_4}$ then $(R_j:n).done \wedge (R_j:n).done'$ and $R_j:n$ will evaluate its $flag$ to $true$. So consider the other case i.e., $Q(v,t,j,s_4)$ is $true \wedge t < j$.

Let s_5 be the state immediately following the event in which read operation $R_j:n$ reads from reader R_t.

Let s_6 be the state immediately following the event in which write operation $W:v+1$ writes to reader R_t for the first time.

By the writer and reader programs and by assumption $s_4 \prec s_5 \prec s_6$. Thus $s_1 \prec s_4 \prec s_5 \prec s_6$ and $Q(v,t,j,s_4)$ is $true$; therefore, by Proposition 4, $Q(v,t,j,s_5)$ is $true$. Therefore, read operation $R_j:n$ will evaluate its $flag$ to $true$. □

Lemma 4 *Let $R_i:k$ and $R_j:n$ be two read operations in proper history h such that read operation $R_i:k$ precedes read operation $R_j:n$ and $j < i$. Assume $R_i:k$ reads its second value of register WR_i from the write operation $W:v+1$ and $R_j:n$ reads its first value of register WR_j from the previous write operation $W:v$. Then read operation $R_i:k$ does not evaluate its $flag$ to $true$.*

Proof: Since the succeeding read operation $R_j:n$ reads its first value of register in register WR_j from an earlier write operation, $(R_i:k).done'$ is $false$. Thus, it is required to show that $\forall u : 0 \leq u < i :: \neg(R_i:k).p_u$. To the contrary, assume that $(R_i:k).p_u$ is $true$ for some u.

Let s_0 be the state immediately following the event in which write operation

$W:v+1$ reads sequence number from reader R_u.

Let s_1 be the state immediately following the event in which write operation $W:v+1$ writes to reader R_0 for the second time.

Let s_2 be the state immediately following the event in which read operation $R_i:k$ writes sequence number to writer.

Let s_3 be the state immediately following the event in which read operation $R_i:k$ reads from reader R_u.

Since it has been assumed that $(R_i:k).p_u$ is *true*, $Q(v+1,u,i,s_3)$ is *true*. By Proposition 2, $s_0 \prec s_2$; by reader program $s_2 \prec s_3$; therefore, $s_0 \prec s_3$. By Proposition 3, s_3 does not occur in the interval $[s_0, s_1]$; therefore, $s_1 \prec s_3$. But then, read operation $R_j:n$ coming after read operation $R_i:k$ cannot read from an earlier write operation $W:v$. This is a contradiction. □

Proposition 1 *Let $W:v$ and $W:v+1$ be two write operations in proper history h. Let s_0 be the state immediately following the event in which write operation $W:v$ reads sequence number from reader R_i. Let s_1 be the state immediately following the event in which write operation $W:v+1$ writes to reader R_i for the first time. Assume $Q(v,i,j,s), i < j$ is true for some state s in the interval $[s_0, s_1]$. Let $R_i:k$ be the read operation that wrote the values appearing in register R_iR_j in state s. If s_2 is the state immediately following the event in which read operation $R_i:k$ writes sequence number to writer then $s_0 \prec s_2$.*

Proof: Assume, to the contrary, that $s_2 \prec s_0$. Then the write operation $W:v$ reads the sequence number written by read operation $R_i:k$,i.e, $(R_i:k).seq_i$, or the sequence number written by read operation $R_i:k+1$,i.e., $(R_i:k+1).seq_i$. Next, we show that $(R_i:k+1).seq_i$ is either $(R_i:k).seq_i$ or $(R_i:k).seq_i \oplus 1$.

A write operation reads the sequence number in register R_iW, increments it by 1, and writes it to the field seq_i of register WR_i. A read operation of R_i reads the sequence number in the field seq_i of register WR_i and writes it to register R_iW. Therefore, $(WR_i.seq_i)_s = (R_iW)_s \vee (WR_i.seq_i)_s = (R_iW)_s \oplus 1$ for all states s in h. In particular, let s be the state immediately following the event in which the read operation $R_i:k+1$ reads from the writer for the first time. In that case, $(WR_i.seq_i)_s =$

$(R_i:k+1).seq_i$ and $(R_iW)_s = (R_i:k).seq_i$. Therefore, $(R_i:k+1).seq_i$ is either $(R_i:k).seq_i$ or $(R_i:k).seq_i \oplus 1$.

Thus, $(W:v).seq_i$ is either $(R_i:k).seq_i \oplus 1$ or $(R_i:k).seq_i \oplus 2$. In either case, $(W:v).seq_i \neq (R_i:k).seq_i$. Therefore, $Q(v,i,j,s)$ is $false$. This is a contradiction. □

Proposition 2 *Let $R_i:k$ be a read operation in proper history h that reads the same set of sequence numbers in both its reads from register WR_i and reads its second value of register WR_i from the write operation $W:v$. Let s_0 be the state immediately following the event in which write operation $W:v$ reads sequence number from reader $R_u, u \leq i$. If s_1 is the state immediately following the event in which read operation $R_i:k$ writes sequence number to writer then $s_0 \prec s_1$.*

Proof: Assume, to the contrary, that $s_1 \prec s_0$. Then write operation $W:v$ reads the sequence number from reader R_i after read operation $R_i:k$ writes sequence number to writer and before read operation $R_i:k+1$ writes sequence number to writer (due to the writer and reader programs). Therefore, $(W:v).seq_i = (R_i:k).seq_i \oplus 1$. Since read operation $R_i:k$ reads its second value of register WR_i from the write operation $W:v$, $(R_i:k).seq_i'$ $= (W:v).seq_i = (R_i:k).seq_i \oplus 1$, which contradicts our assumption that read operation $R_i:k$ read the same set of sequence numbers in both its reads from register WR_i. □

Proposition 3 *Let $W:v$ be a write operation in proper history h and let R_iR_j be any register in the construction. Let s_0 be the state immediately following the event in which this write operation reads sequence number from reader R_i and let s_1 be the state immediately following the event in which this write operation writes to reader R_0 for the second time. Then $Q(v,i,j,s)$ is $false$ for all states s in the interval $[s_0, s_1]$.*

Proof: If $v = 0$ i.e., $W:v$ is the initial write operation, then $Q(0,i,j,s)$ is $false$ for all states s in the interval $[s_0, s_1]$. Otherwise, $v > 0$ and the proof is by induction on i.

Base Case: $i = 0$. Assume, to the contrary, that $Q(v,0,j,s)$ is *true* for some state s in the interval $[s_0, s_1]$. Let $R_0:n$ be the read operation

that wrote the values appearing in register R_0R_j in state s[1]. Assume read operation $R_0{:}n$ read its second value of register WR_0 from the write operation $W{:}u$. Clearly, $u \leq v$. Let s_2 be the state immediately following the event in which read operation $R_0{:}n$ writes sequence number to writer. By Proposition 1, $s_0 \prec s_2$; therefore $u = v-1 \vee u = v$.

If $u = v-1$ then $(R_0{:}n).new' = (W{:}v-1).new$. But, $(W{:}v-1).new \neq (W{:}v).new$. Therefore, $(R_0{:}n).new' \neq (W{:}v).new$. This contradicts the assumption that $Q(v,0,j,s)$ is *true*; therefore $u \neq v-1$.

If $u = v$ then $(R_0{:}n).done'$ will be *false* because $s \prec s_1$. This implies that, $Q(v,0,j,s)$ is *false*; therefore $u \neq v$. This contradicts the assumption that $Q(v,0,j,s)$ is *true*.

Induction Step: Assume the proposition is *true* for all positive integers less than i. It is shown to be true for i. Proof is by contradiction.

Assume that $Q(v,i,j,s)$ is *true* for some state s in the interval $[s_0, s_1]$. Let $R_i{:}n$ be the read operation that wrote the values appearing in register R_iR_j in state s[2]. Assume read operation $R_i{:}n$ read its second value of register WR_i from the write operation $W{:}u$. By reasoning as in the base case it can be shown that $u = v \wedge \neg(R_i{:}n).done'$. Since $Q(v,i,j,s)$ is *true*, there exists $k < i$ such that $(R_i{:}n).p_k$ is *true*.

Let s_2 be the state immediately following the event in which write operation $W{:}v$ read sequence number from reader R_k.

Let s_3 be the state immediately following the event in which read operation $R_i{:}n$ writes sequence number to writer.

Let s_4 be the state immediately following the event in which read operation $R_i{:}n$ reads from reader R_k.

By the writer and reader programs, $s_2 \prec s_0 \prec s_1$ and $s_3 \prec s_4$. By assumption, $s_4 \prec s \prec s_1$ and $Q(v,k,i,s_4)$ is *true*. By induction hypothesis, s_4 does not occur in the interval $[s_2, s_1]$; therefore $s_4 \prec s_2$. Thus, $s_3 \prec s_4 \prec s_2 \prec s_0$. But, by Proposition 1, $s_0 \prec s_3$; this is a contradiction. □

Proposition 4 *Let $W{:}v$ be a write operation in proper history h and let*

[1] If such a read operation $R_0{:}n$ does not exist in h then $(W{:}v).seq_0 = (R_0R_j.seq)_s \oplus 1$ and therefore, $Q(v,0,j,s)$ cannot be *true*.

[2] If such a read operation $R_i{:}n$ does not exist in h then $(W{:}v).seq_i = (R_iR_j.seq)_s \oplus 1$ and therefore, $Q(v,i,j,s)$ cannot be *true*.

R_iR_j be any register in the construction. Let s_0 be the state immediately following the event in which this write operation writes to reader R_0 for the second time and let s_1 be the state immediately following the event in which the next write operation $W{:}v{+}1$ writes to reader R_i for the first time. If $Q(v,i,j,s)$ is true for some state s in the interval $[s_0,s_1]$ then $Q(v,i,j,s')$ is true for all states s' in the interval $[s,s_1]$.

Proof: If $v = 0$ i.e., $W{:}v$ is the initial write operation, then $Q(0,i,j,s)$ is *false* for all states s in the interval $[s_0,s_1]$. Otherwise, $v > 0$ and the proof is by induction on i.

Base Case: $i = 0$. Assume $Q(v,0,j,s)$ is *true* for some state s in the interval $[s_0,s_1]$. Let $R_0{:}n$ be the read operation that wrote the values appearing in register R_0R_j in state s. Assume that read operation $R_0{:}n{+}1$ writes to register R_0R_j in the interval $[s,s_1]$. Let s' be the state immediately following this write event. It required to show that $Q(v,0,j,s')$ is *true*.

Let s_2 be the state immediately following the event in which write operation $W{:}v$ reads sequence number from reader R_0.

Let s_3 be the state immediately following the event in which read operation $R_0{:}n$ writes sequence number to writer.

Let s_4 be the state immediately following the event in which read operation $R_0{:}n$ writes to R_0R_j.

By the writer and reader programs, $s_2 \prec s_0 \prec s_1$ and $s_3 \prec s_4 \prec s'$. By assumption, $s' \prec s_1$ and $Q(v,0,j,s_4)$ is *true*. It is required to show that $Q(v,0,j,s')$ is *true*.

By Proposition 1, $s_2 \prec s_3$; therefore $s_2 \prec s_4$. By Proposition 3, s_4 does not occur in the interval $[s_2,s_0]$; therefore $s_0 \prec s_4$. Thus, the read operation $R_0{:}n{+}1$ reads both its values of register WR_0 from the write operation $W{:}v$ and finds both $(R_0{:}n{+}1).done$ and $(R_0{:}n{+}1).done'$ to be *true*. This implies that $(R_0{:}n{+}1).flag$ is *true*; therefore $Q(v,0,j,s')$ is *true*.

Induction Step: Assume the proposition is *true* for all positive integers less than i. It is shown to be true for i. Assume that $Q(v,i,j,s)$ is *true* for some state s in the interval $[s_0,s_1]$. Let $R_i{:}n$ be the read operation that wrote the values appearing in register R_iR_j in state s. Assume that read operation $R_i{:}n{+}1$ writes to register R_iR_j in the interval $[s,s_1]$. Let s' be

the state immediately following this write event. It is required to show that $Q(v, i, j, s')$ is *true*.

Since $Q(v, i, j, s)$ is *true*, $(R_i{:}n).done' \vee \exists k : 0 \leq k < i :: (R_i{:}n).p_k$. If $(R_i{:}n).done'$ then the proof is analogous to the base case. Consider the other case i.e., $0 \leq k < i \wedge (R_i{:}n).p_k$.

Let s_2 be the state immediately following the event in which read operation $R_0{:}n$ writes sequence number to writer.

Let s_3 be the state immediately following the event in which read operation $R_0{:}n$ reads from reader R_k.

Let s_4 be the state immediately following the event in which read operation $R_0{:}n{+}1$ reads from reader R_k.

Let s_5 be the state immediately following the event in which write operation $W{:}v$ reads sequence number from reader R_k.

Let s_6 be the state immediately following the event in which write operation $W{:}v$ reads sequence number from reader R_i.

By writer and reader programs, $s_5 \prec s_6 \prec s_0 \prec s_1$ and $s_2 \prec s_3 \prec s_4 \prec s'$. By assumption, $s' \prec s_1$ and $Q(v, k, i, s_3)$ is *true*.

It is required to show that $Q(v, i, j, s')$ is *true*. By Proposition 1, $s_6 \prec s_2$; therefore $s_5 \prec s_3$. By Proposition 3, s_3 does not occur in the interval $[s_5, s_0]$; therefore $s_0 \prec s_3$. Thus, $s_0 \prec s_3 \prec s_4 \prec s_1$. $Q(v, k, i, s_3)$ is *true*; therefore, by the induction hypothesis, $Q(v, k, i, s_4)$ is *true*. Therefore, $Q(v, i, j, s')$ is *true*. □

Proposition 5 *Let $W{:}v$ be a write operation in proper history h and let R_iR_j be any register in the construction. Let s_0 be the state immediately following the event in which this write operation writes to reader R_0 for the second time and let s_1 be the state immediately following the event in which the next write operation $W{:}v{+}1$ writes to reader R_i for the first time. If $Q(v, i, j, s)$ is true for some state s in the interval $[s_0, s_1)$ then $\forall k < j :: [(WR_k.done)_s \vee \exists p < k :: Q(v, p, k, s)$ is true$]$.*

Proof: If $v = 0$ i.e., $W{:}v$ is the initial write operation, then $Q(0, i, j, s)$ is $false$ for all states s in the interval $[s_0, s_1]$. Otherwise, $v > 0$ and the proof is by induction on i.

Base Case: $i = 0$. Assume $Q(v, 0, j, s)$ is *true* for some state s in the

interval $[s_0, s_1)$. Let $R_0{:}n$ be the read operation that wrote the values appearing in register R_0R_j in state s. Let $k < j$. It is required to show that $Q(v, 0, k, s)$ is *true*.

Let s_2 be the state immediately following the event in which write operation $W{:}v$ reads sequence number from R_0.

Let s_3 be the state immediately following the event in which read operation $R_0{:}n$ writes sequence number to writer.

Let s_4 be the state immediately following the event in which read operation $R_0{:}n$ writes to reader R_k.

Let s_5 be the state immediately following the event in which read operation $R_0{:}n$ writes to reader R_j.

By reader and writer programs, $s_2 \prec s_0 \prec s_1$ and $s_3 \prec s_4 \prec s_5$. By assumption $s_5 \preceq s \prec s_1$ and $Q(v, 0, k, s_4)$ is *true*.

By Proposition 1, $s_2 \prec s_3$; therefore $s_2 \prec s_4$. By Proposition 3, s_4 does not occur in the interval $[s_2, s_0]$; therefore $s_0 \prec s_4$. Thus $s_0 \prec s_4 \prec s \prec s_1$. $Q(v, 0, k, s_4)$ is *true*; therefore, by Proposition 4, $Q(v, 0, k, s)$ is *true*.

Induction Step: Assume the proposition is *true* for all positive integers less than i. It is shown to be true for i. Assume that $Q(v, i, j, s)$ is *true* for some state s in the interval $[s_0, s_1)$. Let $k < j$. It is required to show that $(WR_k.done)_s \vee \exists p < k :: Q(v, p, k, s)$ is *true*.

If $i < k < j$ then the proof is analogous to the base case. The two remaining cases, $i = k < j$ and $k < i < j$, are considered separately.

Case 1: $(i = k < j)$ Let $R_i{:}n$ be the read operation that wrote the values appearing in register R_iR_j in state s. Therefore, $(R_i{:}n).done' \vee \exists u < i :: (R_i{:}n).p_u$

Let s_2 be the state immediately following the event in which write operation $W{:}v$ reads sequence number from reader R_i.

Let s_3 be the state immediately following the event in which read operation $R_i{:}n$ writes sequence number to writer.

Let s_4 be the state immediately following the event in which read operation $R_i{:}n$ writes to reader R_j.

By writer and reader programs, $s_2 \prec s_0 \prec s_1$ and $s_3 \prec s_4$. By assumption, $s_4 \preceq s \prec s_1$ and $Q(v, i, j, s_4)$ is *true*.

Consider the first possibility i.e., $(R_i{:}n).done'$ is $true$. By Proposition 1, $s_2 \prec s_3$; therefore, $R_i{:}n$ reads its second value of register WR_i from the write operation $W{:}v-1$ or from the write operation $W{:}v$. But, if $R_i{:}n$ reads its second value of register WR_i from $W{:}v-1$ then $Q(v,i,j,s)$ cannot be $true$. Therefore, $R_i{:}n$ must read its second value of register WR_i from the write operation $W{:}v$. In that case, since $R_i{:}n$ found $(WR_i.done)$ to be $true$, $(WR_i.done)_s$ will also be $true$.

Now, consider the other possibility i.e., $u < i \wedge (R_i{:}n).p_u$.

Let s_5 be the state immediately following the event in which read operation $R_i{:}n$ reads from reader R_u.

Let s_6 be the state immediately following the event in which write operation $W{:}v$ reads sequence number from reader R_u.

By writer and reader programs, $s_6 \prec s_2$ and $s_3 \prec s_5$. By assumption, $Q(v,u,i,s_5)$ is $true$.

By Proposition 1, $s_2 \prec s_3$; thus $s_6 \prec s_2 \prec s_3 \prec s_5$. By Proposition 3, s_5 does not occur in the interval $[s_6, s_0]$; therefore $s_0 \prec s_5$. Since $Q(v,u,i,s_5)$ is $true$, by Proposition 4, $Q(v,u,i,s)$ is $true$.

Case 2: $(k < i < j)$ Let $R_i{:}n$ be the read operation that wrote the values appearing in register R_iR_j in state s. Therefore, $(R_i{:}n).done' \vee \exists u < i :: (R_i{:}n).p_u$

If $(R_i{:}n).done'$ then, by reasoning as in Case 1, it can be shown that $(WR_i.done)_s$ is $true$; therefore, by writer program, $(WR_k.done)_s$ is $true$.

Consider the other possibility i.e., $u < i \wedge (R_i{:}n).p_u$. Define states s_2, s_3, s_4, s_5, s_6 as in Case 1. By writer and reader programs, $s_6 \prec s_2 \prec s_0 \prec s_1$ and $s_3 \prec s_5 \prec s_4$. By assumption, $s_4 \preceq s \prec s_1$, $Q(v,i,j,s_4)$ is $true$, and $Q(v,u,i,s_5)$ is $true$.

By reasoning as in Case 1, it can be shown that $s_0 \prec s_5$. Therefore, by the induction hypothesis, $(WR_k.done)_{s_5} \vee \exists t < k :: Q(v,t,k,s_5)$.

If $(WR_k.done)_{s_5}$ is $true$ then, due to the precedence $s_0 \prec s_5 \prec s \prec s_1$, $(WR_k.done)_s$ is $true$. Consider the other possibility i.e., $t < k \wedge Q(v,t,k,s_5)$ is $true$. $s_0 \prec s_5 \prec s \prec s_1$ and the write operation $W{:}v+1$ writes to reader R_i before writing to R_t; therefore, by Proposition 4, $Q(v,t,k,s)$ is $true$. □

References

[ASG 86] Anderson, J., Singh, A., and Gouda, M., "The Elusive Atomic Register," *Technical Report TR.86.29*, Department of Computer Sciences, University of Texas at Austin, 1986.

[KKV 86] Kirousis, L.M., Kranakis, E., and Vitanyi, P., "Atomic Multi-reader Register," detailed abstract, 1986.

[La 86] Lamport, L., "On Interprocess Communication, Parts I and II," *Distributed Computing*, Vol. 1, pp. 77-101, 1986.

[Mi 86] Misra, J., "Axioms for Memory Access in Asynchronous Hardware Systems," *ACM Transactions on Programming Languages and Systems*, Vol. 8, pp 142-153, 1986.

[Pe 83] Peterson, G. L., "Concurrent Reading while Writing," *ACM Transactions on Programming Languages and Systems*, Vol. 5, pp. 46-55, 1983.

[Pe 86] Peterson, G.L., *private communication*, 1986.

[VA 86] Vitanyi, P., and Awerbuch, B., "Atomic Shared Register Access by Asynchronous Hardware," FOCS, 1986.