THE ELUSIVE ATOMIC REGISTER

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

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

TR-86-29 December 1986

Abstract

We present a construction of a 1-writer/m-reader atomic register using 1-writer/1-reader atomic registers. This construction solves an open problem suggested by Lamport, and closes the only remaining gap in the chain of constructions that implements a k-writer/m-reader/n-bit atomic register (the most sophisticated) using 1-writer/1-reader/1-bit safe registers (the most primitive).

---

*Work supported by NSF grant no. ECS 83-04734

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 cornerstone 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 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, we adopt the following notation 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.)

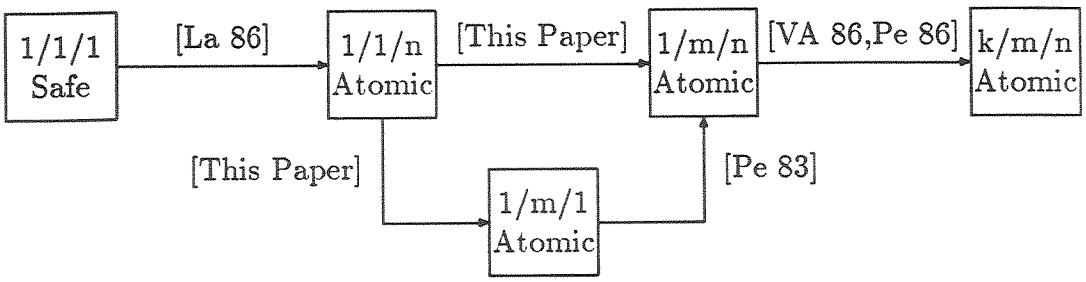

Figure 1: Two Chains of Register Constructions.

The only missing construction in either chain is the construction of a 1/m atomic register from 1/1 atomic registers. This is the subject of this paper. Previously, this problem has been mentioned by Lamport [La 86], and Vitanyi and Awerbuch [VA 86]. In a related work, Misra [Mi 86] has presented a consistent and complete set of axioms that guarantee atomicity without giving any constructions to realize them.

The rest of this paper is organized as follows. In Section 2, we formally define the problem of constructing 1/m atomic registers from 1/1 atomic registers Then, in Section 3, we present a construction of a 1/2 atomic register from 1/1 atomic registers. In Section 4, we extend this construction to a 1/m atomic register for any m. Concluding remarks appear in Section 5.

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 i^{th} 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 1/2 Atomic Register

In the following sections, we show how to construct a 1/2 atomic register from 1/1 atomic registers. In Sections 3.1 and 3.2, we describe the architecture of the construction and its initial state. In Section 3.3 we define the programs for the writer and each reader. Lastly, in Section 3.4, we prove that the construction is atomic.

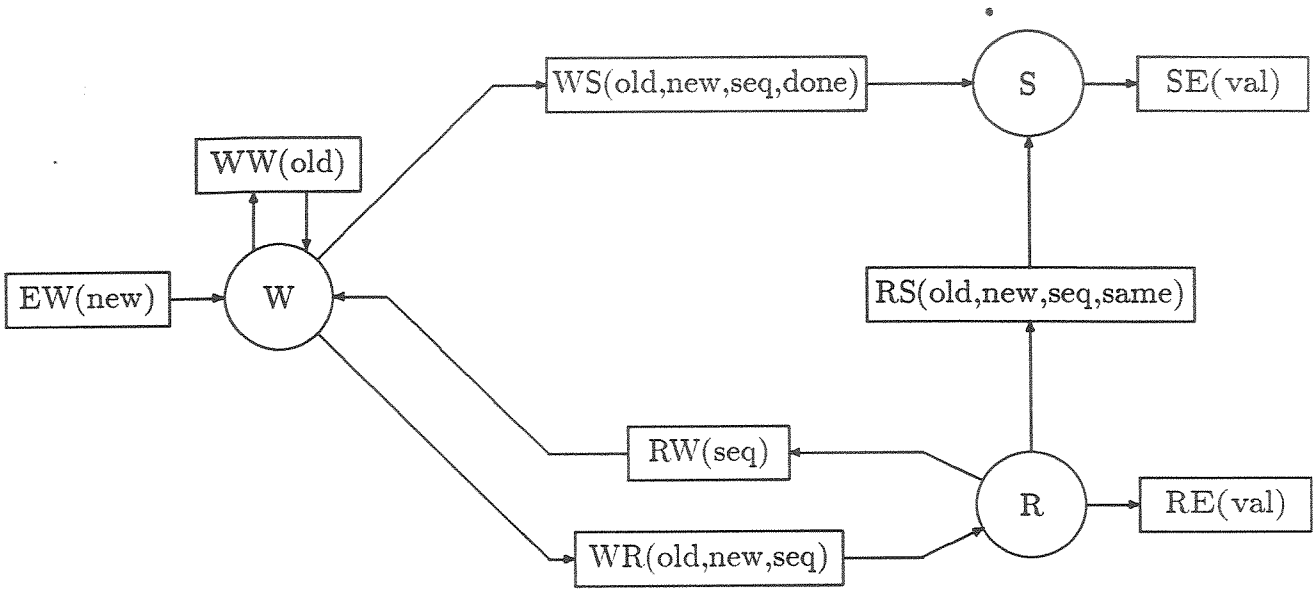

Figure 2: 1/2 Atomic Register Architecture.

3.1 Architecture

The architecture is illustrated in Figure 2. In this figure, programs are represented by circles, and registers are represented by boxes. An arrow is drawn from a program to a register to indicate that the program writes to the register; an arrow is drawn from a register to a program to indicate that the program reads from the register. For convenience, the writer program is called W, and the two reader programs are called R and S.

Our construction has five internal registers, denoted WW, WS, WR, RW, and RS, one input register, denoted EW, and two output registers, denoted RE and SE. Each register contains one or more fields. In Figure 2, register fields are listed in parentheses after the register name. For example, the register WR has fields *old*, *new*, and *seq*.

Informally, the register fields are used in the following way. The register fields *old*, *new*, and *val* have the same type as the register being constructed. Basically, *old* contains the value stored in the register before the most recent

write operation; *new* contains the value written by the most recent write operation; and *val* contains the last value read by either R or S.

The register fields *done* and *same* are both boolean. During a write operation, W writes twice to WS; *done* is set to *true* during the second write to indicate that the write operation is complete. During a read operation by R, WR is read twice; R sets *same* to *true* iff it reads the same values for *old, new,* and *seq* both times.

The register field *seq* stores a sequence number in the range 0..2.

3.2 Initial State

The initial register values are as follows:

$$\begin{array}{ll} \mathrm{EW} = (y), & \mathrm{WW} = (x), \\ \mathrm{WR} = (y, x, 0), & \mathrm{WS} = (y, x, 0, \mathit{true}), \\ \mathrm{RS} = (y, x, 0, \mathit{true}), & \mathrm{RW} = (0), \\ \mathrm{RE} = (x), \text{ and} & \mathrm{SE} = (x), \end{array}$$

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

3.3 Algorithm

In this section, we describe the algorithm informally. The algorithm is specified by defining the body for each program. The bodies for programs W, R, and S are shown in Figures 3, 4, and 5, respectively.

Before describing the algorithm in detail, it is worth pointing out two constraints imposed by Definition 10. First of all, the condition of *safety* implies that a read operation that does not overlap a write operation must return the most recently written value. Second, suppose that two read operations, say r_1 and r_2, overlap the same write operation and that r_1 precedes r_2. Then, the the condition of *precedence* implies that r_2 must return the new value if r_1 does. These constraints should be kept in mind when reading the description that follows.

During a write operation, W writes the triple *(old value, new value, sequence number)* to both WR and WS. WR is written to once. WS is written to twice, the first time with the variable *done* set to *false* and the

```
0: read (new) from EW;
1: read (old) from WW;
2: if old ≠ new then
        3: read (seq) from RW;
        4: seq := (seq + 1) mod 3;
        5: write (old, new, seq, false) to WS;
        6: write (old, new, seq) to WR;
        7: write (old, new, seq, true) to WS;
        8: write (new) to WW;
   fi
```

Figure 3: <body> of Writer W.

```
 9: read (old1, new1, seq1) from WR;
10: write (seq1) to RW;
11: read (old2, new2, seq2) from WR;
12: if old1 ≠ old2 ∨ new1 ≠ new2 ∨ seq1 ≠ seq2 then
        13: same := false;
    else
        14: same := true;
    fi
15: write (old1, new1, seq1, same) to RS;
16: write (new1) to RE;
```

Figure 4: <body> of Reader R.

```
17: read (old1, new1, seq1, done1) from WS;
18: read (old, new, seq, same) from RS;
19: read (old2, new2, seq2, done2) from WS;
20: flag := (old1 = old = old2 ∧ new1 = new = new2∧
            seq1 = seq = seq2 ∧ done1 = done2 ∧ same);
21: if done2 ∨ flag then
        22: val := new2;
    else
        23: val := old2;
    fi
24: write (val) to SE;
```

Figure 5: <body> of Reader S.

second time with *done* set to *true*. W computes its sequence number by reading the sequence number in RW and then incrementing it. Whenever R performs a read operation, it copies the sequence number from WR to RW. Therefore, if the sequence numbers in registers WS and RS are different, then WS was written to after RS. S uses this information to determine if a read by R has occurred recently.

R reads WR twice, and sets the variable *same* to *true* if it reads the same values both times. Between these reads, R writes the sequence number from the first read to RW. Since W writes to WR only once, R must return one of the new values that it reads in order to ensure safety. As seen in Figure 4, R returns the new value from the first read. Suppose that R overlaps a write operation, and that there was a preceding read by S that returned the new value. Then, the algorithm ensures that the write operation in question has written to WR before R begins. This is because S returns the new value only if either $done_2$ or $flag$ is *true*. If $done_2$ is *true*, then WR has certainly been written to. If *flag* is *true*, then there was a read by R that preceded S and returned the new value, and, therefore, WR has been written to (Strictly speaking, it is possible for *flag* to be set to true as a consequence of a read by R that just finished, but returned a value read in the past. However, we will ignore this possibility for now.) Therefore, R must return the new value.

S reads WS twice and always returns either the new or old value from the second read. In between these reads, S reads from RS. The information from these three reads is used to compute the boolean variable *flag* in statement 20. It can be shown that if S overlaps a write operation and there was a preceding read by R that returned the new value, then *flag* will be *true*. Also, if S does not overlap a write operation, then $done_2$ will be *true*. Therefore, if either $done_2$ or *flag* is *true*, then S returns the new value. Otherwise, if $done_2$ and *flag* are both *false*, then S returns the old value.

3.4 Proof of Correctness

3.4.1 Notation

In this section, we describe notation that will be used in the proofs in the following section and in the Appendix.

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

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.

3.4.2 Proofs

We now prove that our construction is atomic by defining a function ϕ for each proper history of the construction, and then showing that ϕ meets the three conditions *integrity*, *safety*, and *precedence* stated in Definition 10.

Suppose that $R{:}\,m{:}\,9$[1] reads from $W{:}i$, $S{:}\,n{:}\,17$ reads from $W{:}j$, and $S{:}\,n{:}\,19$ reads from $W{:}k$. Then, ϕ is defined as follows.

$$\begin{aligned} \phi(R{:}m) &= i \\ \phi(S{:}n) &= \begin{cases} k & \text{if } (S{:}\,n).done_2 \\ k-1 & \text{if } \neg(S{:}\,n).done_2 \wedge \neg(S{:}\,n).flag \\ j & \text{if } \neg(S{:}\,n).done_2 \wedge (S{:}\,n).flag \end{cases} \end{aligned}$$

Proof of Integrity: We prove that for any read operations $R{:}m$ and $S{:}n$, $(R{:}\,m).new_1 = (W{:}\,\phi(R{:}m)).new$ and $(S{:}\,n).val = (W{:}\,\phi(S{:}n)).new$.

Suppose that $R{:}\,m{:}\,9$ reads from $W{:}i$. Then,

$$\begin{aligned} (R{:}\,m).new_1 &= (W{:}\,i).new && \text{since } R{:}\,m{:}\,9 \text{ reads from } W{:}i \\ &= (W{:}\,\phi(R{:}m)).new && \text{by the definition of } \phi \end{aligned}$$

Suppose that $S{:}\,n{:}\,17$ reads from $W{:}j$ and $S{:}\,n{:}\,19$ reads from $W{:}k$. In order to show that $(S{:}\,n).val = (W{:}\,\phi(S{:}n)).new$, we must consider

[1] When referring to events, statement numbers are used instead of the actual statements themselves.

three possibilities. First, suppose that $\phi(S{:}n) = k$. Then,

$$\begin{array}{lcll} (S{:}\,n).val & = & (S{:}\,n).new_2 & \text{by the algorithm for reader } S \\ & = & (W{:}\,k).new & \text{since } S{:}\,n{:}\,19 \text{ reads from } W{:}k \\ & = & (W{:}\,\phi(S{:}n)).new & \text{by the definition of } \phi \end{array}$$

Second, suppose that $\phi(S{:}n) = k - 1$. Then,

$$\begin{array}{lcll} (S{:}\,n).val & = & (S{:}\,n).old_2 & \text{by the algorithm for reader } S \\ & = & (W{:}\,k).old & \text{since } S{:}\,n{:}\,19 \text{ reads from } W{:}k \\ & = & (W{:}\,k{-}1).new & \text{by the algorithm for writer } W \\ & = & (W{:}\,\phi(S{:}n)).new & \text{by the definition of } \phi \end{array}$$

Third, suppose that $\phi(S{:}n) = j$. Note that $\phi(S{:}n) = j \Rightarrow (S{:}\,n).flag$. Therefore,

$$\begin{array}{lcll} (S{:}\,n).val & = & (S{:}\,n).new_2 & \text{by the algorithm for reader } S \\ & = & (S{:}\,n).new_1 & \text{since } (S{:}\,n).flag \text{ holds} \\ & = & (W{:}\,j).new & \text{since } S{:}\,n{:}\,17 \text{ reads from } W{:}j \\ & = & (W{:}\,\phi(S{:}n)).new & \text{by the definition of } \phi \end{array}$$

This completes the proof of integrity. □

Proof of Safety: Let $R{:}m$ be any read operation for reader R. By the definition of ϕ, $R{:}\,m{:}\,9$ reads from $W{:}\,\phi(R{:}m){:}\,9$. Therefore, $R{:}m$ does not precede $W{:}\phi(R{:}m)$, and $W{:}\phi(R{:}m){+}1$ does not precede $R{:}m$.

Let $S{:}n$ be any read operation for reader S. We show that $S{:}n$ does not precede $W{:}\phi(S{:}n)$, and that $W{:}\phi(S{:}n){+}1$ does not precede $S{:}n$. Suppose that $S{:}\,n{:}\,17$ reads from $W{:}j$, and $S{:}\,n{:}\,19$ reads from $W{:}k$. By the definition of ϕ, $\phi(S{:}n) \leq k$. Therefore, since $S{:}\,n{:}\,19$ reads from $W{:}k$, $S{:}n$ does not precede $W{:}\phi(S{:}n)$.

We now show that $W{:}\phi(S{:}n){+}1$ does not precede $S{:}n$. Suppose, to the contrary, that $W{:}\phi(S{:}n){+}1$ precedes $S{:}n$. Then, since $S{:}\,n{:}\,17$ reads from $W{:}j$, and since $S{:}\,n{:}\,19$ reads from $W{:}k$, $\phi(S{:}n){+}1 \leq j \leq k$. Thus, by the definition of ϕ, $j = k$ and $\phi(S{:}n) = k - 1$. Hence, $(S{:}\,n).done_2$ is *false*. Therefore, $S{:}\,n{:}\,17$ and $S{:}\,n{:}\,19$ both read from $W{:}\,\phi(S{:}n){+}1{:}\,5$. This implies that $W{:}\phi(S{:}n){+}1$ does not precede $S{:}n$; contradiction. This completes the proof of safety. □

Proof of Precedence: Let r and r' be two read operations in h such that r precedes r'. In order to prove precedence, we must show that $\phi(r) \leq \phi(r')$. There are four cases to consider:

Case 1: $r = S{:}m$ and $r' = R{:}n$

Case 2: $r = S{:}m$ and $r' = S{:}n$

Case 3: $r = R{:}m$ and $r' = S{:}n$

Case 4: $r = R{:}m$ and $r' = R{:}n$

The proofs for these four cases use Lemmas 1, 2, and 3, which are stated and proved in the Appendix.

Case 1: Suppose that $S{:}m$ precedes $R{:}n$. Furthermore, suppose that $S{:}\,m{:}\,17$ reads from $W{:}i$, $S{:}\,m{:}\,19$ reads from $W{:}j$, $R{:}\,n{:}\,9$ reads from $W{:}k$, and $R{:}\,n{:}\,11$ reads from $W{:}l$. By the algorithms for R and S, $i \leq j$ and $k \leq l$. Also, since W writes to WS before WR, $k \geq j - 1$.

By the definition of ϕ, $\phi(R{:}n) = k$ and $\phi(S{:}m) \leq j$. Therefore, if $k \geq j$, the result holds. So, assume that $k = j - 1$.

Since $R{:}\,n{:}\,9$ reads from $W{:}j{-}1$, $S{:}\,m{:}\,19$ must read from $W{:}\,5{:}\,j$. Therefore, $(S{:}\,m).done_2$ is *false*. By the definition of ϕ, this implies that either $\phi(S{:}m) = i$ or $\phi(S{:}m) = j - 1$. If $\phi(S{:}m) = j - 1$, then $\phi(S{:}m) = \phi(R{:}n)$. Otherwise, $\phi(S{:}m) = i$, and, by the definition of ϕ, $(S{:}\,m).flag$ is *true*. Then, by Lemma 1, $\phi(S{:}m) = i < j = \phi(R{:}n)$. □

Case 2: Suppose that $S{:}m$ precedes $S{:}n$. Furthermore, suppose that $S{:}\,m{:}\,17$ reads from $W{:}i$, $S{:}\,m{:}\,19$ reads from $W{:}j$, $S{:}\,m{:}\,17$ reads from $W{:}k$, and $S{:}\,m{:}\,19$ reads from $W{:}l$. By the algorithm for reader S, and since $S{:}m$ precedes $S{:}n$, $i \leq j \leq k \leq l$.

Assume that $j < l$. By the definition of ϕ, $\phi(S{:}m) \leq j$, and $\phi(S{:}n) \geq min(k, l-1)$. Therefore, $\phi(S{:}m) \leq \phi(S{:}n)$. For the rest of the proof, assume $j = k = l$.

If $(S{:}\,n).done_2$ is *true*, then $\phi(S{:}m) \leq j = \phi(S{:}n)$. If $(S{:}\,m).done_2$ is *true*, then $(S{:}\,n).done_2$ is also *true*, and $\phi(S{:}m) = j = \phi(S{:}n)$. Consider the remaining possibility; i.e., $(S{:}\,m).done_2$ and $(S{:}\,n).done_2$ are both *false*.

If $(S{:}\,m).flag$ is *false*, then $\phi(S{:}m) = j - 1 \leq \phi(S{:}n)$.* Otherwise, $(S{:}\,m).flag$ is *true* and $\phi(S{:}m) = i$. If $i = j$, then $S{:}\,m{:}\,17$, $S{:}\,m{:}\,19$, $S{:}\,n{:}\,17$, and $S{:}\,n{:}\,19$ all read from $W{:}\,i{:}\,5$. Therefore, by Lemma 2, $(S{:}\,n).flag$ is also *true* and $\phi(S{:}m) = i = \phi(S{:}n)$. Otherwise, $i < j$, and $\phi(S{:}m) = i \leq j - 1 \leq \phi(S{:}n)$. □

Case 3: Suppose that $R{:}m$ precedes $S{:}n$. Furthermore, suppose that $R{:}\,m{:}\,9$ reads from $W{:}i$, $R{:}\,m{:}\,11$ reads from $W{:}j$, $S{:}\,n{:}\,17$ reads from $W{:}k$, and $S{:}\,n{:}\,19$ reads from $W{:}l$. By the algorithm for reader R, $i \leq j$, and by the algorithm for reader S, $k \leq l$. Also, since $R{:}m$ precedes $S{:}n$, and since W writes to WS before WR, $j \leq k$. Therefore, $i \leq j \leq k \leq l$.

If $i < l$, then $\phi(R{:}m) = i \leq min(k, l{-}1) \leq \phi(S{:}n)$. Also, If $(S{:}\,n).done_2$ is *true*, then $\phi(R{:}m) = i \leq l = \phi(S{:}n)$. Therefore, we are left with only one case; namely, $i = j = k = l$ and $(S{:}\,n).done_2 = false$. In this case, $S{:}\,n{:}\,17$ and $S{:}\,n{:}\,19$ both read from $W{:}\,i{:}\,5$, and $R{:}\,m{:}\,9$ and $R{:}\,m{:}\,11$ both read from $W{:}\,i{:}\,6$. Therefore, by Lemma 3, $(S{:}\,n).flag$ is *true*. Hence, by the definition of ϕ, $\phi(S{:}n)$ is equal to either k or l. Therefore, $\phi(R{:}m) = \phi(S{:}n)$. □

Case 4: Suppose that $R{:}m$ precedes $R{:}n$. Furthermore, suppose that $R{:}\,m{:}\,9$ reads from $W{:}i$, and $R{:}\,n{:}\,9$ reads from $W{:}j$. Then, since $R{:}m$ precedes $R{:}n$, $i \leq j$. Therefore, by the definition of ϕ, $\phi(R{:}m) = i \leq j = \phi(R{:}n)$. □

4 Construction of a 1/m Atomic Register

We now show how to extend the construction given in the previous section to $m > 2$ readers. Our construction relies on the following observation: since reader S does not write to any internal register, we can replace it by $(m-1)$ readers $S_0, \ldots, S_{m-2}$, as depicted in Figure 6, without introducing any multiple writer registers. Registers WS and RS now become 1/(m - 1) atomic registers, which can be constructed recursively using 1/(m - 2) atomic registers.

The body of program $S_i, i = 0, \ldots, (m-2)$, is the same as the body

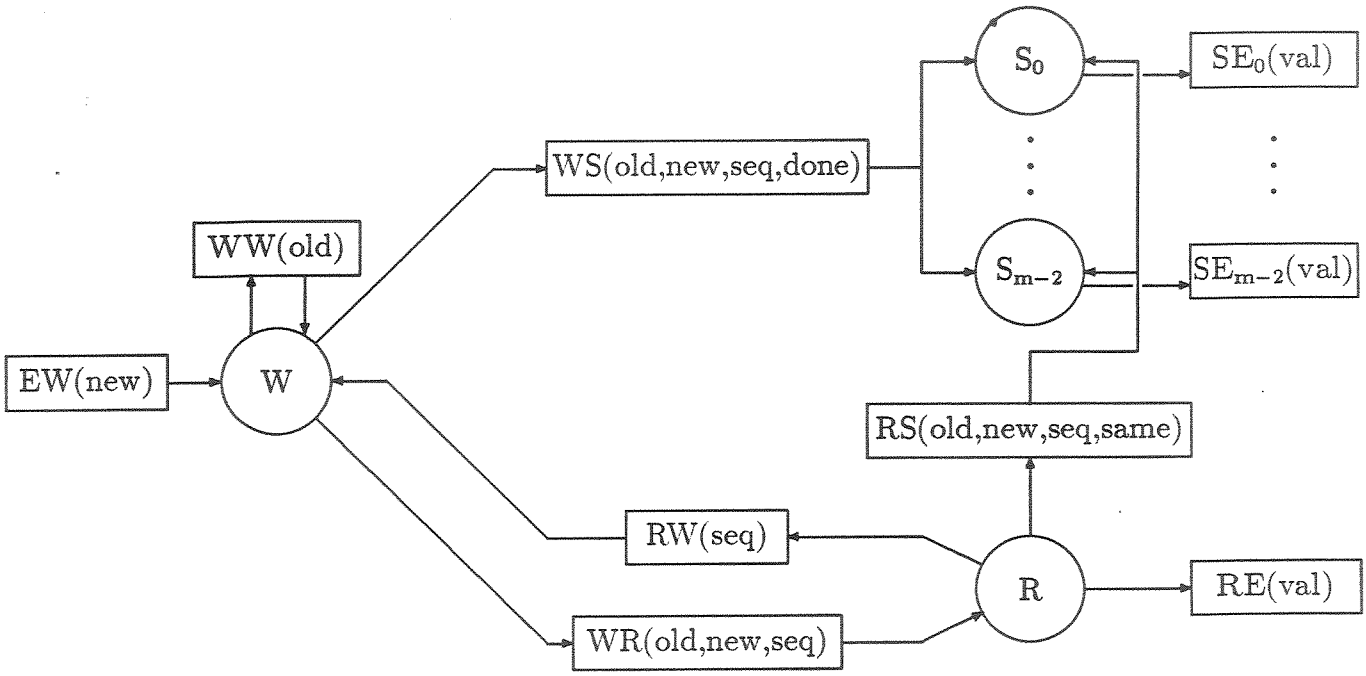

Figure 6: 1/m Atomic Register Architecture.

for S depicted in Figure 5, except that SE is replaced by SE_i in statement 24. The bodies of programs W and R are the same as in the original construction. The proof of correctness for this construction is essentially the same as that given for the two reader construction.

Let us now examine the complexity of our construction. The metric we will use is the number of 1/1 atomic bits used in the construction. Let $B(m,n)$ denote the number of 1/1 atomic bits required to construct a 1/m/n atomic register. Then, referring to Figure 6, we have the following.

- Register WW uses n 1/1 atomic bits.
- Register WS uses $B(m-1, 2n+3)$ 1/1 atomic bits if $m > 2$, and $(2n+3)$ 1/1 atomic bits if $m = 2$.
- Register RS uses $B(m-1, 2n+3)$ 1/1 atomic bits if $m > 2$, and

$(2n+3)$ 1/1 atomic bits if $m = 2$.

- Register RW uses 2 1/1 atomic bits.
- Register WR uses $(2n+2)$ 1/1 atomic bits.

Therefore, $B(m,n)$ can be defined as follows.

$$B(m,n) = \begin{cases} 2B(m-1, 2n+3) + 3n + 4 & \text{if } m > 2 \\ 7n + 10 & \text{if } m = 2 \end{cases}$$

The solution to this equation is

$$B(m,n) = 2^{2m-1}(n+3) - 2^{m+2} - n + 2.$$

That is, it takes $O(n4^m)$ 1/1 atomic bits to build a 1/m/n atomic register.

5 Discussion

We have shown that it is possible to construct a 1/m atomic register from 1/1 atomic registers. With this result, we now know that it is possible to build a k/m/n atomic register from 1/1/1 safe registers. Previously, it was not known whether such a construction existed.

The construction that we have presented is relatively simple in the two reader case. Also, the two reader construction can elegantly be extended to $m > 2$ readers through a recursive construction. Therefore, to understand how the construction works, it is necessary only to understand the two reader case. However, this simplicity comes at a high price. Namely, the number of 1/1 atomic bits required to construct a 1/m atomic register is exponential in m. In [SAG 86] we show how to extend the two reader construction to a 1/m/n construction that requires only $O(m^2n)$ 1/1 atomic bits.

After developing the constructions in [SAG 86], 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 $O(m^3+m^2n)$ which is greater than the space complexity of the construction presented in [SAG 86].

Appendix: Lemmas and Propositions

In this section, we state and prove three lemmas and one proposition. The three lemmas are used in the proof of correctness in Section 3.4. The proposition is a safety property that is used in the proofs of the first two lemmas.

The following notation is adopted in the proofs that follow:

$P(s)$	denotes the predicate $[(WS.new)_s \neq (WR.new)_s] \Rightarrow$ $\neg[(WS.old)_s = (RS.old)_s \wedge (WS.new)_s = (RS.new)_s \wedge$ $(WS.seq)_s = (RS.seq)_s \wedge (RS.same)_s]$.
$Q(s,i)$	denotes the predicate $(S\colon i).old = (RS.old)_s \wedge$ $(S\colon i).new = (RS.new)_s \wedge (S\colon i).seq = (RS.seq)_s \wedge$ $(S\colon i).same = (RS.same)_s$.
$V(s,i)$	denotes the predicate $(S\colon i).old_1 = (S\colon i).old_2 = (WS.old)_s \wedge$ $(S\colon i).new_1 = (S\colon i).new_2 = (WS.new)_s \wedge$ $(S\colon i).seq_1 = (S\colon i).seq_2 = (WS.seq)_{s_1}$.
$s_0 \prec s_1$	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.
$\ominus$	denotes modulo 3 subtraction.

Let h be any proper history of the construction. Without loss of generality, we assume that for every write operation w in h, $(w).old \neq (w).new$. The following discussion is in the context of h.

In order to avoid special cases in the proofs, we assume that the *hypothetical* operations $W\colon -1$, $R\colon -1$, and $S\colon -1$ occur "before" the intial state. Furthermore, we assume that $W\colon -1$ precedes $R\colon -1$, and that $R\colon -1$ precedes $S\colon -1$. To be consistent with the initial state defined in Section 3.2, we

assume that $(W\colon -1).old = y$, $(W\colon -1).new = -1$, and $(W\colon \dot{-}1).seq = 0$.

Lemma 1 *Suppose that $S{:}m$, $R{:}n$, $W{:}k$, and $W{:}k{+}1$ are operations in h, and that $S{:}m$ precedes $R{:}n$. Furthermore, suppose that $S\colon m\colon 17$ and $S\colon m\colon 19$ both read from $W\colon k{+}1\colon 5$, and that $R\colon n\colon 9$ reads from $W\colon k\colon 6$. Then, $(S\colon m).flag$ must be false.*

Proof: Let s_0 denote the state following the occurrence of $W\colon k{+}1\colon 5$, let s_1 denote the state following the occurrence of $S\colon m\colon 18$, let s_2 denote the state following the occurrence of $R\colon n\colon 9$, and let s_3 denote the state following the occurrence of $W\colon k{+}1\colon 6$.

Since $S{:}m$ precedes $R{:}n$, $s_1 \prec s_2$. Since $S\colon m\colon 17$ reads from $W\colon k{+}1\colon 5$, $s_0 \prec s_1$. Since $R\colon n\colon 9$ reads from $W\colon k\colon 6$, $s_2 \prec s_3$. Therefore, $s_0 \prec s_1 \prec s_3$.

By the algorithm for reader S, $Q(s_1, m)$ must be *true*. Also, since $S\colon m\colon 17$ and $S\colon m\colon 19$ both read from $W\colon k{+}1\colon 5$, and since $s_0 \prec s_1 \prec s_3$, $V(s_1, m)$ must be *true*.

Since $s_0 \prec s_1 \prec s_3$, $(WS.new)_{s_1} \neq (WR.new)_{s_1}$. Therefore, by Proposition 1, we have $\neg[(WS.old)_{s_1} = (RS.old)_{s_1} \wedge (WS.new)_{s_1} = (RS.new)_{s_1} \wedge (WS.seq)_{s_1} = (RS.seq)_{s_1} \wedge (RS.same)_{s_1}]$. This implies that $(S\colon m).flag$ is *false*. □

Lemma 2 *Suppose that $S{:}m$, $S{:}n$, and $W{:}k$ are operations in h, and that $S{:}m$ precedes $S{:}n$. Furthermore, suppose that $S\colon m\colon 17$, $S\colon m\colon 19$, $S\colon n\colon 17$, and $S\colon n\colon 19$ all read from $W\colon k\colon 5$. Then $(S\colon m).flag \Rightarrow (S\colon n).flag$.*

Proof: Let us assume that $(S\colon m).flag$ is *true*. Then, we must prove that $(S\colon n).flag$ is also *true*.

Since $S\colon m\colon 17$, $S\colon m\colon 19$, $S\colon n\colon 17$, and $S\colon n\colon 19$ all read from $W\colon k\colon 5$, it is required to prove that $(S\colon m).old = (S\colon n).old$, $(S\colon m).new = (S\colon n).new$, $(S\colon m).seq = (S\colon n).seq$, and $(S\colon m).same = (S\colon n).same$.

Assume that $S\colon m\colon 18$ reads from $R\colon i\colon 15$, and that $S\colon n\colon 18$ reads from $R\colon j\colon 15$. Then, since $S{:}m$ precedes $S{:}n$, $i \leq j$.

If $R\colon i\colon 9$, $R\colon i\colon 11$, $R\colon j\colon 9$, and $R\colon j\colon 11$ all read from the same write to WR, then $(S\colon n).flag$ must be *true*. We now prove that this *must* be the case.

Let s_0 denote the state following the occurrence of $W\colon k\colon 5$, let s_1 denote the state following the occurrence of $W\colon k\colon 6$, and let s_2 denote the state following the occurrence of $W\colon k\colon 7$.

Let s_3 denote the state following the occurrence of $S\colon m\colon 18$ and let s_4 denote the state following the occurrence of $S\colon n\colon 18$. Since $S\colon m$ precedes $S\colon n$, $s_3 \prec s_4$. Since $S\colon m\colon 17$ reads from $W\colon k\colon 5$, $s_0 \prec s_3$. Since $S\colon n\colon 19$ reads from $W\colon k\colon 5$, $s_4 \prec s_2$. Therefore, $s_0 \prec s_3 \prec s_4 \prec s_2$.

Let s_5 denote the state following the occurrence of $R\colon i\colon 15$. Since $S\colon m\colon 18$ reads from $R\colon i\colon 15$, $s_5 \prec s_3$.

We now claim that $s_0 \prec s_5$. This can be justified as follows. Assume to the contrary that $s_5 \prec s_0$. Recall that $s_0 \prec s_3$. Therefore, $s_5 \prec s_0 \prec s_3$. Since $S\colon m\colon 18$ reads from $R\colon i\colon 15$, $Q(s_0, m)$ must be *true*. Since $S\colon m\colon 17$ and $S\colon m\colon 19$ read from $W\colon k\colon 5$, $V(s_0, m)$ must be *true*. Therefore, since $(S\colon m).flag$ is *true*, $[(WS.old)_{s_0} = (RS.old)_{s_0} \wedge (WS.new)_{s_0} = (RS.new)_{s_0} \wedge (WS.seq)_{s_0} = (RS.seq)_{s_0} \wedge (RS.same)_{s_0}]$ is *true*. However, this contradicts Proposition 1, since $(WS.new)_{s_0} \neq (WR.new)_{s_0}$.

We have thus far established that $s_0 \prec s_5 \prec s_3 \prec s_4 \prec s_2$. We can also show that $s_1 \prec s_5$ as follows. Since $S\colon m\colon 18$ reads from $R\colon i\colon 15$, $Q(s_5, m)$ must be *true*. As noted previously, $V(s_0, m)$ must be *true*. Since $s_0 \prec s_5 \prec s_2$, $(WS.old)_{s_5} = (WS.old)_{s_0}$, $(WS.new)_{s_5} = (WS.new)_{s_0}$, and $(WS.seq)_{s_5} = (WS.seq)_{s_0}$. Therefore, since $(S\colon m).flag$ is *true*, $[(WS.old)_{s_5} = (RS.old)_{s_5} \wedge (WS.new)_{s_5} = (RS.new)_{s_5} \wedge (WS.seq)_{s_5} = (RS.seq)_{s_5} \wedge (RS.same)_{s_5}]$ is *true*. Therefore, by Proposition 1, $(WS.new)_{s_5} = (WR.new)_{s_5}$. This implies that $s_1 \prec s_5$. Therefore, we have established that $s_0 \prec s_1 \prec s_5 \prec s_3 \prec s_4 \prec s_2$.

Let s_6 denote the state following the occurrence of $R\colon j\colon 11$. Note that $i \leq j \Rightarrow s_5 \prec s_6$. Also, since $S\colon n\colon 18$ reads from $R\colon j\colon 15$, $s_6 \prec s_4$. Therefore, $s_0 \prec s_1 \prec s_5 \prec s_6 \prec s_4 \prec s_2$. This implies that $R\colon i\colon 9$, $R\colon i\colon 11$, $R\colon j\colon 9$, and $R\colon j\colon 11$ all read from $W\colon k\colon 6$. As noted above, this implies that $(S\colon n).flag$ is *true*. □

Lemma 3 *Suppose that $R\colon m$, $S\colon n$, and $W\colon k$ are operations in h, and that $R\colon m$ precedes $S\colon n$. Furthermore, suppose that $R\colon m\colon 9$ and $R\colon m\colon 11$ both read from $W\colon k\colon 6$, and that $S\colon n\colon 17$ and $S\colon n\colon 19$ both read from $W\colon k\colon 5$. Then, $(S\colon n).flag$ must be true.*

Proof: Since $S\colon n\colon 17$ and $S\colon n\colon 19$ both read from $W\colon k\colon 5$, it is required to prove that $(S\colon n).old = (W\colon k).old$, $(S\colon n).new = (W\colon k).new$, $(S\colon n).seq = (W\colon k).seq$, and $(S\colon n).same = true$.

Let s_0 denote the state following the occurrence of $W\colon k\colon 5$, let s_1 denote the state following the occurrence of $W\colon k\colon 6$, and let s_2 denote the state following the occurrence of $W\colon k\colon 7$.

Let s_3 denote the state following the occurrence of $R\colon m\colon 9$, let s_4 denote the state following the occurrence of $S\colon n\colon 18$, and let s_5 denote the state following the occurrence of $S\colon n\colon 19$. Since $R\colon m\colon 9$ reads from $W\colon k\colon 6$, $s_1 \prec s_3$. Since $S\colon n\colon 19$ reads from $W\colon k\colon 5$, $s_5 \prec s_2$. Since $R\colon m$ precedes $S\colon n$, $s_3 \prec s_4$. Therefore, $s_0 \prec s_1 \prec s_3 \prec s_4 \prec s_5 \prec s_2$.

Suppose that $S\colon n\colon 18$ reads from $R\colon i\colon 15$. Let s_6 denote the state following the occurrence of $R\colon i\colon 9$, let s_7 denote the state following the occurrence of $R\colon i\colon 11$, and let s_8 denote the state following the occurrence of $R\colon i\colon 15$.

Since $S\colon n\colon 18$ reads from $R\colon i\colon 15$, and since $R\colon m$ precedes $S\colon n$, $i \geq m$. Therefore, $s_3 \preceq s_6$. The fact that $S\colon n\colon 18$ reads from $R\colon i\colon 15$ also implies that $s_8 \prec s_4$. Thus, we have established that $s_0 \prec s_1 \prec s_3 \preceq s_6 \prec s_7 \prec s_8 \prec s_4 \prec s_5 \prec s_2$. Therefore, $R\colon i\colon 9$ and $R\colon i\colon 11$ both read from $W\colon k\colon 6$. Since $S\colon n\colon 18$ reads from $R\colon i\colon 15$, this implies that $(S\colon n).old = (W\colon k).old$, $(S\colon n).new = (W\colon k).new$, $(S\colon n).seq = (W\colon k).seq$, and $(S\colon n).same = true$. □

Proposition 1 *$P(s)$ is true for each state s occurring in h.*

Proof: Assume to the contrary that $\neg P(s)$ holds for some state s in h. Note that $\neg P(s) = (WS.new)_s \neq (WR.old)_s \wedge (WS.old)_s = (RS.old)_s \wedge (WS.new)_s = (RS.new)_s \wedge (WS.seq)_s = (RS.seq)_s \wedge (RS.same)_s$.

Let $W\colon n$ denote the last write operation to write to WS before state s. Let $R\colon m$ denote the last read by R to write to RS before state s.

Suppose that $(W\colon n).old = u$, $(W\colon n).new = v$, and $(W\colon n).seq = q$. Then, $\neg P(s) \Rightarrow (R\colon m).old_1 = u, (R\colon m).new_1 = v, (R\colon m).seq_1 = q$, and $(R\colon m).same = true$.

Assume that $R\colon m\colon 9$ reads from $W\colon i\colon 6$ and $R\colon m\colon 11$ reads from $W\colon j\colon 6$, where $i \leq j$. Let s_0 denote the state following the occurrence

of $W\colon n\colon 6$. Then, the first conjunct of $\neg P(s)$ implies that $s \prec s_0$. Therefore, since $R\colon m\colon 11$ reads from $W\colon j\colon 6$ and since $R\colon m\colon 15$ is the last write to RW before state s, $j < n$.

Since $(R\colon m).same = true$, $(W\colon i).old = (W\colon j).old = u$, $(W\colon i).new = (W\colon j).new = v$, and $(W\colon i).seq = (W\colon j).seq = q$.

Since $(W\colon j).old = (W\colon n).old = u$ and $(W\colon j).new = (W\colon n).new = v$, and since $j < n$, $W\colon j{+}1$ exists and $(j+1) < n$.

Let s_1 denote the state following the occurrence of $R\colon m\colon 11$, and let s_2 denote the state following the occurrence of $W\colon j{+}1\colon 6$. Since $R\colon m\colon 11$ reads from $W\colon j\colon 6$, $s_1 \prec s_2$.

Let s_3 denote the state following the occurrence of $R\colon m\colon 10$, and let s_4 denote the state following the occurrence of $W\colon n\colon 3$. Note that $s_3 \prec s_1$. Also, since $(j+1) < n$, $s_2 \prec s_4$. Therefore, since $s_1 \prec s_2$, it follows that $s_3 \prec s_4$. This implies that $W\colon n\colon 3$ cannot read from $R\colon k\colon 10$, where $k < m$.

Suppose that $R\colon m{+}1\colon 15$ occurs in h, and let s_5 denote the state following the occurrence of this event. Since $R\colon m\colon 15$ is the last write to RW before state s, $s \prec s_5$. This implies that $W\colon n\colon 3$ cannot read from $R\colon k\colon 10$, where $k > m+1$. Therefore, we have shown that $W\colon n\colon 3$ must read from either $R\colon m\colon 10$ or $R\colon m{+}1\colon 10$.

Suppose that $W\colon n\colon 3$ reads from $R\colon m\colon 10$. Then, since $(R\colon m).seq_1 = q$, $(W\colon n).seq = q \oplus 1$. But, this contradicts the fact that $(W\colon n).seq = q$.

Suppose that $W\colon n\colon 3$ reads from $R\colon m{+}1\colon 10$. Since $(W\colon n).seq = q$, it follows that $(R\colon m{+}1).seq_1 = q \ominus 1 = q \oplus 2$. Note that a write operation reads the sequence number in register RW, increments it by 1, and writes it to the field *seq* of register WR. Also, a read operation of R reads the sequence number in the field *seq* of register WR and writes it to register RW. Therefore, $(WR.seq)_{s'} = (RW.seq)_{s'} \vee (WR.seq)_{s'} = (RW.seq)_{s'} \oplus 1$ for all states s' in h. In particular, let s_6 be the state immediately following the occurrence of $R\colon m{+}1\colon 9$. In that case, $(WR.seq)_{s_6} = (R\colon m{+}1).seq_1$ and $(RW.seq)_{s_6} = (R\colon m).seq_1$. Therefore, $(R\colon m{+}1).seq_1$ is either $(R\colon m).seq_1$ or $(R\colon m).seq_1 \oplus 1$. That is, $(R\colon m{+}1).seq_1 \neq q \oplus 2$; contradiction. □

References

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

[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.

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

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