The Elusive Atomic Register

(Updated Version)

Ambuj K. Singh[*] James H. Anderson[†]
Mohamed G. Gouda[†]

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

TR-89-14 May 1989

Abstract

We present two constructions of a multiple-reader atomic register from single-reader atomic registers. The first is a recursive construction; a two-reader construction defines the base step, and a scheme to construct an M-reader register from $(M-1)$-reader registers defines the induction step. This construction, although simple to understand and verify, has exponential complexity. Our second construction is also an extension of the above two-reader construction. This construction, while more complicated, has optimal complexity; it requires $O(M^2 + MN)$ atomic single-reader bits for an N-bit register.

Keywords: atomicity, atomic register, interleaving semantics, shared variable

CR Categories: D.4.1, D.4.2, F.3.1

[*]Work supported in part by NSF Grant ECS 83-04734 and Office of Naval Research Contract N00014-86-K-0182.
[†]Work supported in part by Office of Naval Research Contract N00014-86-K-0763.

1 Introduction

The currently accepted theory of concurrent computing is deeply rooted in the concept of atomic registers. An *atomic register* is a data object that is read or written by one or more processes according to the following assumption: If several read or write operations of the register are enabled simultaneously in different processes, then these operations are executed in some 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 by different processes — and to then show that an atomic register can be constructed using these registers. Such a construction consists of a set of "internal" registers, along with some programs that access (read or write) them. A process reads or writes the constructed atomic register by invoking one of these programs. Different programs can be invoked by different processes concurrently; the net effect, however, resembles that of a serial invocation. The programs are restricted to be wait-free, i.e., synchronization primitives, such as P, V, or **await**, and unbounded busy-wait loops are not allowed. This restriction guarantees that a process reads or writes the constructed register in a finite amount of time, regardless of the activities of other processes. (This also means that the read or write of a process is immune to the failure of other processes that also access the register.) The wait-freedom restriction distinguishes the problem of constructing an atomic register from the classic readers-writers problem [5].

Peterson [11] was the first to suggest the problem of constructing atomic registers from safe registers. A *safe register* is a data object that can be read and written concurrently by different processes; if a read operation overlaps a write operation, then it 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 smaller steps. Figure 1 depicts two chains of register constructions that lead from single-writer, single-reader, single-bit safe registers to K-writer, M-reader, N-bit atomic registers. The notation $K/M/N$ denotes a register that can be written by K processes, read by M processes, and store an N-bit value. Each step in the figure is labeled by a reference to the paper(s) in which the given construction is presented.

Henceforth, we will concern ourselves only with single-writer atomic registers. The problem of constructing a multiple-reader atomic register from single-reader atomic registers was mentioned as an open problem by Lamport [8] and Vitanyi and Awerbuch [14]. The first solution to the problem was presented in [2]. In this solution, we first defined a two-reader construction, and then constructed an M-reader register recursively from $(M-1)$-reader registers. This solution, though easy to explain and understand, uses an exponential number

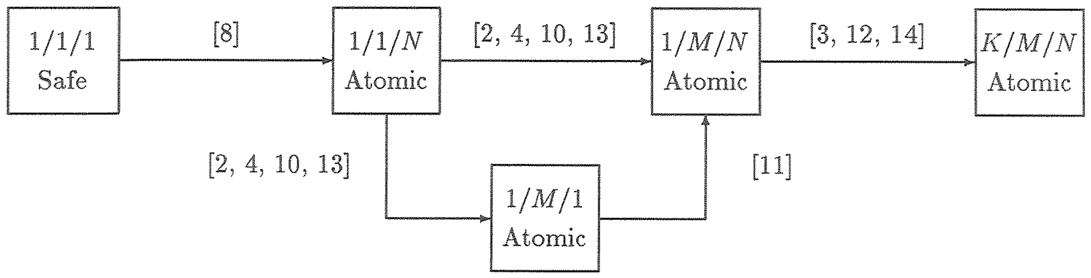

Figure 1: Two Chains of Register Constructions.

of single-reader atomic registers. Subsequently, several solutions with polynomial complexity have been presented [4, 10] including our polynomial construction [13]. This construction has polynomial complexity, and is a generalization of our two-reader construction.

In this paper, we present two constructions, one based upon the solution in [2] and another based upon the solution in [13]. The first construction is presented because of its simplicity. By examining this construction, the reader should be able to gain some insight as to why the second, more complicated construction is correct. The second construction and its correctness proof are the main focus of this paper. The presentation of this solution differs from that of [13] in two respects. First, the solution presented here is of optimal complexity, whereas the one given in [13] is not. (Actually, an optimal solution can be attained by combining the solution in [13] with constructions by Lamport [8] and Peterson [11] — see [13] for details.) Second, the correctness proof presented in this paper is more rigorous and formal (and, we hope, easier to understand) than the proof presented in [13].

The rest of the paper is organized as follows. In Section 2 we formally define the problem of constructing an M-reader atomic register from single-reader atomic registers. In Section 3 we construct a two-reader register from single-reader registers and extend this construction recursively to obtain an M-reader construction. In Section 4 we present our polynomial construction. In Section 5 we prove that our polynomial construction is correct; the proof makes use of several lemmas and propositions, which are stated and proved in an appendix. Concluding remarks appear in Section 6.

2 Register Construction

Register constructions can be defined in a number of different ways. Our choice of definitions is based on simplicity and convenience.

Terminology: In order to avoid confusion, we henceforth capitalize terms such as "Read" and "Write" when they apply to the *constructed* register, and leave them uncapitalized when they apply to the internal shared variables of a construction. □

We view the Writer and each Reader of a construction as a program that is invoked by a process in order to Write (Read) a value to (from) the register. The program for the Writer has one input parameter indicating the value to be Written; similarly, the program for each Reader has one output parameter indicating the value Returned. As an example, see the constructions depicted in Figures 3 and 5.

Each variable of a construction is read by at most one program — this restriction arises since our aim is to construct a multiple-reader register from single-reader registers. We also require that all variables are bounded in size. (There is a very simple solution if the variables are unbounded [14].) As mentioned in the introduction, each program of a construction is "wait-free," i.e., synchronization primitives and busy-wait loops are not allowed. (For a more formal definition of wait-freedom, refer to [1].) Because each shared variable corresponds to a single-writer, single-reader atomic register, a statement of a program can either read a single shared variable, or write a single shared variable, but not both; i.e., in each statement, there is at most one occurrence of a shared variable.

Next, we define several concepts that are needed to state the correctness condition for a multiple-reader construction. These definitions apply to a given construction.

Definition: A *state* is an assignment of values to the variables of the construction. One state is designated as the *initial state*. □

Definition: An *event* is an execution of a statement of a program. □

Definition: Let t and u be any two states of a construction such that state u is the result of executing some statement at state t. If e is the event corresponding to this statement execution, then we write $t \xrightarrow{e} u$. A *history* of a construction is a sequence $t_0 \xrightarrow{e_0} t_1 \xrightarrow{e_1} \cdots$ where t_0 is the initial state. □

Definition: Event e *precedes* another event f in a history iff e occurs before f in the history. □

Definition: The set of events in a history corresponding to some complete program execution is called an *operation*. An operation p *precedes* another operation q in a history iff each event of p precedes all events of q. □

Observe that the precedes relation is a total order on events and a partial order on operations.

For the proof of correctness of a construction, it is sufficient to consider only histories in which an initial Write operation precedes all other operations, and that do not contain any incomplete program executions (i.e., operations). From now on, we deal only with such histories.

Notation: We denote the *ith* operation of the Writer, where $i \geq 0$, by $W\!:\!i$. (Thus, $W\!:\!0$ denotes the initial Write.) □

Following Lamport [8], we define the correctness condition for a construction as follows.

Definition: Let h be any history of a construction. h is said to be *atomic* iff there exists a function ϕ that maps every Read operation in h to some natural number i, where $W\!:\!i$ is a Write operation in h, such that the following three conditions hold.

- *Integrity:* For each Read operation r in h, the value Returned by r is the same as the value Written by $W\!:\!\phi(r)$.
- *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.
- *Precedence:* For any two Read operations r and s in h, if r precedes s then $\phi(r) \leq \phi(s)$. □

Definition: A register construction is *correct* iff all its histories are atomic. □

3 Recursive Construction

In this section, we first give a construction for a two-reader register, and then generalize this construction to define an M-reader construction that uses $(M-1)$-reader registers. The two constructions together show how an M-reader register can be constructed from single-reader registers for any M.

3.1 Two-Reader Construction

The architecture of the construction is depicted in Figure 2. In this figure, boxes denote programs, circles denote variables, and arrows denote direction of communication; an outgoing arrow from a program to a variable indicates that the program writes the variable, while an arrow in the reverse direction indicates that the program reads the variable. The Writer is called W, and the two Readers are denoted R and S. The construction uses four single-reader shared variables, which we denote WR, WS, RW, and RS. Notice that program S does not write any shared variable — this fact is crucial, and is exploited in the next subsection to recursively define an M-reader register from $(M-1)$-reader registers.

The shared variable declarations along with the programs W, R, and S are depicted in Figure 3. The field names appearing in the **type** definitions are as follows.

alt: A bit that alternates in value with each operation of W.

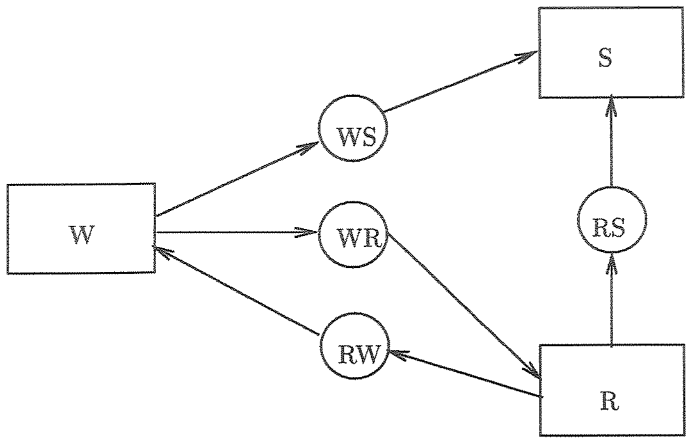

Figure 2: Architecture of the two-reader construction.

done: A bit that distinguishes the two values written by an operation of W to variable WS.

flag: A bit that indicates whether both values read by an operation of R from variable WR are the same.

new: The "current" value of the constructed register. (*valtype* is the type of the constructed register.)

old: The "previous" value of the constructed register.

seq: A modulo-3 integer "sequence number." ($\oplus$ denotes modulo-3 addition.)

In the programs W, R, and S, we use a special syntax in order to distinguish reads and writes of shared variables from reads and writes of local variables. A program reads a given shared variable Y by executing a statement of the form "**read** x **from** Y," where x is a local variable of the same type as Y. A program writes a shared variable Y by executing a statement of the form "**write** x **to** Y." If variable Y consists of m fields, then x is an m-tuple; the *ith* component of x is a local variable whose value is to be stored in the *ith* field of Y. We use similar names (sometimes identical) for the components of x and the fields of Y, so the correspondence should be obvious.

As indicated in Figure 3, any state in which $RS.flag$ is false is a suitable initial state. (But, recall that each history, by assumption, begins with the Write operation $W\!:\!0$.)

The two-reader construction is included in order to give the reader (of this paper) some insight as to why the more-complicated polynomial construction described in Section 4 is correct. We do not wish to be rigorous at this point, and therefore a formal correctness proof has been omitted (refer to [2]). Instead, we now provide a rather brief, intuitive discussion of our proof obligations of integrity, safety, and precedence.

```
type $WRtype$ = record   $new : valtype$;   $seq : 0..2$;   $alt$ : boolean   end;
     $WStype$ = record   $old, new : valtype$;   $seq : 0..2$;   $alt, done$ : boolean   end;
     $RStype$ = record   $seq : 0..2$;   $alt$ : boolean;   $flag$ : boolean   end
var  $WR : WRtype$;
     $WS : WStype$;
     $RW : 0..2$;
     $RS : RStype$
initialization
     $\neg RS.flag$
```

```
program $W(val : valtype)$
var    $old, new : valtype$;
       $alt$ : boolean;
       $q, seq : 0..2$
begin
       $old,\ new,\ alt := new,\ val,\ \neg alt$;
       read $q$ from $RW$;
       $seq := q \oplus 1$;
       write $(old,\ new,\ seq,\ alt,\ false)$ to $WS$;
       write $(new,\ seq,\ alt)$ to $WR$;
       write $(old,\ new,\ seq,\ alt,\ true)$ to $WS$
end
```

```
program $R$ returns $valtype$
var    $x, y : WRtype$;
       $flag$ : boolean
begin
       read $x$ from $WR$;
       write $x.seq$ to $RW$;
       read $y$ from $WR$;
       $flag := x = y$;
       write $(flag,\ x.seq,\ x.alt)$ to $RS$;
       return$(x.new)$
end
```

```
program $S$ returns $valtype$
var    $x, y : WStype$;
       $v : RStype$;
begin
       read $x$ from $WS$;
       read $v$ from $RS$;
       read $y$ from $WS$;
       if $y.done \lor (x = y\ \land\ v.flag\ \land$
                   $x.seq = v.seq\ \land\ x.alt = v.alt)$ then
          return$(y.new)$
       else
          return$(y.old)$
       fi
end
```

Figure 3: Two-reader construction.

To prove that integrity holds, we are required to show that the value Returned by a given Read operation is, in fact, Written by some Write operation. To prove that safety holds, we must show that this Write operation either "overlaps" the Read operation or precedes it. In the latter case, we have the additional proof obligation that no "later" Write operation precedes the given Read operation.

As we shall see in Section 5, designing a construction that satisfies integrity and safety is straightforward — in fact, integrity and safety turn out to be an almost immediate consequence of our definition of ϕ. The difficulty lies in satisfying precedence. Proving precedence amounts to showing that no "new-then-old" conflicts occur; that is, if Read operation r precedes Read operation r', then r' does not Return a value that is "older" than the value Returned by r.

Since all variables of a construction are required to be single-reader, the Writer cannot simultaneously make the new value available to both Readers. Instead, it must write the new value to each Reader separately, which gives rise to potential new-then-old conflicts. In our construction, W writes to S first — so, we must make sure that S Returns the new value only when this value is already available to R.

Suppose that s is an operation of S, and that both values read by s from register WS are written by Write operation w. By the program for S, s Returns $y.new$ only if $y.done$ is true or if $x = y$, $v.flag$ is true, $x.seq = v.seq$, and $x.alt = v.alt$. In the former case, the second write by w to WS occurs before s reads from WS for the second time; thus, w writes to WR before the occurrence of any subsequent operation of Reader R. In the latter case, it can be shown that w writes to WR before s reads from RS (see Lemma 3 in the appendix); thus, in this case as well, w writes to WR before the occurrence of any subsequent operation of Reader R. Therefore, new-then-old conflicts do not occur.

3.2 M-Reader Construction Using $(M-1)$-Reader Registers

In the two-reader construction presented above, S does not write any shared variable. This suggests a recursive construction for an M-reader register in which S is replaced by $M-1$ Readers, and WS and RS are each replaced by an $(M-1)$-reader register. This construction is illustrated in Figure 4. Each register has the same fields as before, but now registers WS and RS are read by the $M-1$ programs $S_1, \ldots, S_{M-1}$.

In order to construct an M-reader register, we apply the above scheme $M-1$ times. The correctness of the construction follows from the correctness of the two-reader construction. Unfortunately, there is a price to pay for this simplicity; the construction of a multiple-reader register requires a number of bits that is exponential in the number of Readers [2].

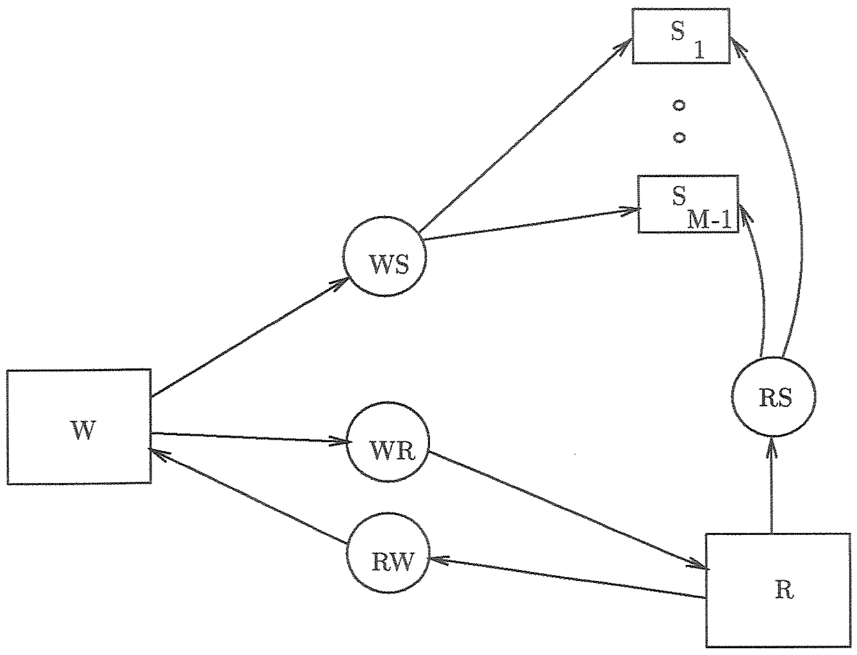

Figure 4: Architecture of the M-reader construction.

4 Polynomial Construction

Our second construction is depicted in Figure 5. Each shared variable is of the single-reader kind. Variable $WR[i]$ is written by the Writer and read by Reader i. Variable $RW[i]$ is written by Reader i and read by the Writer. Variable $RR[i, j]$ is written by Reader i and read by Reader j, where $i < j$. Any state in which each $RR[i, j].flag$ is false for each i and j, where $1 \leq i < j \leq M$, is a suitable initial state.

The Writer is a generalization of the Writer of the two-reader algorithm given in Figure 3. Instead of a single sequence number, we now have M different sequence numbers; each sequence number is like a "token" that is circulated between the Writer and a specific Reader. The Writer writes to the Readers in two passes; in the first pass the Writer writes to the Readers in the order from M to 1, and in the second pass this order is reversed. The value that the Writer writes to each Reader is the same as the value written by W to S in the two-reader case, except that all M sequence numbers are included.

The Reader is obtained by "combining" the two Readers R and S of the two-reader algorithm. Each Reader reads two values from the Writer, x and y. Between these reads, its sequence number is written back to the Writer and a value is read from each lower numbered Reader. The values read from the Writer and other Readers are used to compute the $flag$ bit, which indicates whether the old or new value from y is to be Returned. Note that $flag$ is assigned a value based upon the expressions $p_0, \ldots, p_{i-1}$. These expressions have been

```
type $WRtype$ = record  $old, new : valtype$;  $seq$ : array[1..$M$] of 0..2;  $alt, done$ : boolean  end;
     $RRtype$ = record  $seq$ : 0..2;  $alt, flag$ : boolean  end
var  $WR$ : array[1..$M$] of $WRtype$;
     $RW$ : array[1..$M$] of 0..2;
     $RR[i]$ : array[$i$+1..$M$] of $RRtype$ for each $i$ in the range $1 \leq i < M$
initialization
     $(\forall i, j : 1 \leq i < j \leq M : \neg RR[i,j].flag)$

program $Writer(val : valtype)$
var   $old, new : valtype$;
      $alt$ : boolean;
      $q, seq$ : array[1..$M$] of $RWtype$;
      $k : 1..M$
begin
  0:  $old,\ new,\ alt := new,\ val,\ \neg alt$;
  1:  for $k = 1$ to $M$ do read $q[k]$ from $RW[k]$ od;
  2:  for $k = 1$ to $M$ do $seq[k] := q[k] \oplus 1$ od;
  3:  for $k = M$ to 1 do write ($old,\ new,\ seq[1..M],\ alt,\ false$) to $WR[k]$ od;
  4:  for $k = 1$ to $M$ do write ($old,\ new,\ seq[1..M],\ alt,\ true$) to $WR[k]$ od
end

program $Reader(i : 1..M)$ returns $valtype$
var   $x, y : WRtype$;
      $v$ : array[$1..i-1$] of $RRtype$;
      $flag$ : boolean;
      $k$ : integer
always
      $(\ p_0\ \equiv\ y.done\ \wedge\ x.seq[i] = y.seq[i]\ )\ \wedge$
      $(\ \forall k : 0 < k < i : p_k\ \equiv\ x = y\ \wedge\ v[k].flag \wedge\ x.seq[k] = v[k].seq\ \wedge\ x.alt = v[k].alt\ )$
begin
  0:  read $x$ from $WR[i]$;
  1:  write $x.seq[i]$ to $RW[i]$;
  2:  for $k = 1$ to $i - 1$ do read $v[k]$ from $RR[k, i]$ od;
  3:  read $y$ from $WR[i]$;
  4:  $flag := (\exists k : 0 \leq k < i : p_k)$;
  5:  for $k = i + 1$ to $M$ do write ($flag,\ y.seq[i],\ y.alt$) to $RR[i, k]$ od;
  6:  if $flag$ then return($y.new$) else return($y.old$) fi
end
```

Figure 5: Polynomial multiple-reader construction.

introduced as a shorthand, and are defined in the **always** section. Before Returning a value, the Reader writes a value to each higher numbered Reader. This value is the same as the value written by R to S in the two-reader construction, except that we have $y.seq[i]$ instead of $y.seq$.

The number of single-reader atomic shared bits used in our construction is as follows:

- $WR[i]$, $1 \leq i \leq M$, uses $2M + 2N + 2$ bits.
- $RW[i]$, $1 \leq i \leq M$, uses 2 bits.
- $RR[i, j]$, $1 \leq i < j \leq M$, uses 4 bits.

Thus, the complexity of our construction is $6M^2 + 2MN$. It is well known that the lower bound for this problem is $O(M^2 + MN)$ bits [7]; thus, our construction is optimal.

5 Proof of Correctness

We prove that our construction is correct by defining a function ϕ for a given history, and by showing that the defined ϕ meets the three conditions of integrity, safety, and precedence defined in Section 2. The following notations and definitions are used in the proof.

Notation: Let p be an operation, and x be any local variable of p. Then, $p!x$ denotes the final value of variable x as assigned by operation p.

In order to avoid using too many parentheses, we define a binding order for the symbols that we use. The following is a list of these symbols, grouped by binding power; the groups are ordered from highest binding power to lowest.

$[\,]$, $(\,)$

$.$

$!$

$\neg$, $:$

$+$, $-$, $\oplus$

$=$, $\neq$, $<$, $>$, $\leq$, $\geq$, $\prec$, $\preceq$

$\wedge$, $\vee$

$\Rightarrow$, $\equiv$

Let p be an operation of program P and i be a label of a statement in P. If i is not a **for** loop, then $p{:}i$ denotes the event corresponding to the execution of statement i in operation p. Otherwise, if i is a **for** loop, then $p{:}i.j$ denotes the event corresponding to the iteration of the loop in which the loop counter equals j. □

Examples: Let w be a Write operation, and r be an operation of Reader i. The following events are referred to repeatedly in the proof.

$w:1.i$	The Writer reads from $RW[i]$.
$w:3.i$	The Writer writes to $WR[i]$ for the first time.
$w:4.i$	The Writer writes to $WR[i]$ for the second time.
$r:0$	Reader i reads from $WR[i]$ for the first time.
$r:1$	Reader i writes to $RW[i]$.
$r:2.j$	Reader i reads from $RR[j,i]$, where $j < i$.
$r:3$	Reader i reads from $WR[i]$ for the second time.
$r:5.j$	Reader i writes to $RR[i,j]$, where $j > i$.

□

Notation: Let e and f be two events in some history. Then, $e \prec f \equiv e$ precedes f, and $e \preceq f \equiv (e \prec f) \vee (e = f)$. □

Definition: Let e be the event corresponding to the execution of the statement **read** x **from** Y in an operation p, where x is a local variable and Y is a shared variable. If the value that p reads from Y is written by operation q, then we say that operation q *determines* $p!x$.

□

Observation: Let r and s be any two operations of Reader i and Reader j respectively such that $r!y$ is determined by the Write operation $W:m$ and $s!y$ is determined by the Write operation $W:n$. If r precedes s and $m > n$ then $m = n+1$ and $i > j$.

Proof:

$W:m$ determines $r!y$

$=$ {Definition of determines}

$(W:m):3.i \prec r:3$

$=$ $\{r$ precedes $s\}$

$(W:m):3.i \prec r:3 \prec s:3$

$=$ $\{W:n$ determines $s!y$ and definition of determines$\}$

$(W:m):3.i \prec r:3 \prec s:3 \prec (W:n{+}1):3.j$

$=$ {Transitivity of $\prec$}

$(W:m):3.i \prec (W:n{+}1):3.j$

$=$ {Write operations occur sequentially in a history}

$m \leq n+1 \ \wedge \ (W:m):3.i \prec (W:n{+}1):3.j$

$=$ $\{m > n\}$

$m = n+1 \ \wedge \ (W:m):3.i \prec (W:n{+}1):3.j$

$=$ {Predicate calculus}

$m = n+1 \;\wedge\; (W\!:\!m)\!:\!3.i \prec (W\!:\!m)\!:\!3.j$

$=$ {Program for Reader}

$m = n+1 \;\wedge\; i > j$ □

Definition: Let r be any Read operation, and suppose that Write operation $W\!:\!m$ determines $r!y$. Then, $\phi(r)$ is defined as follows.

$$\phi(r) = \begin{cases} m & \text{if } r!\mathit{flag} \\ m-1 & \text{otherwise} \end{cases}$$

□

Proof of Integrity: Let r be any Read operation, and suppose that Write operation $W\!:\!m$ determines $r!y$. If $\phi(r) = m$, then $r!\mathit{flag}$ is true (definition of ϕ) and r Returns the value $r!y.\mathit{new}$, i.e., the value Written by $W\!:\!m$. If $\phi(r) = m-1$, then $r!\mathit{flag}$ is false (definition of ϕ) and r Returns the value $r!y.\mathit{old}$, i.e., the value Written by $W\!:\!m-1$. □

Proof of Safety: Let r be any Read operation, and suppose that Write operation $W\!:\!m$ determines $r!y$. If $\phi(r) = m$, then because $W\!:\!m$ determines $r!y$, r does not precede $W\!:\!m$ and $W\!:\!m+1$ does not precede r. If $\phi(r) = m-1$, then $r!\mathit{flag}$ is false. Hence, $r!y.\mathit{done}$ is also false, which implies that r does not precede $W\!:\!m-1$ and $W\!:\!m$ does not precede r. □

Proof of Precedence: The proof of precedence is quite complicated and consists of a somewhat lengthy case analysis. Most of the cases that must be considered are proved as propositions and lemmas in the appendix.

Let r be any operation of Reader i, and s be any operation of Reader j such that r precedes s. Our proof obligation is $\phi(r) \le \phi(s)$.

Assume that Write operations $W\!:\!m$ and $W\!:\!n$ determine $r!y$ and $s\!:\!y$ respectively. Observe the following.

$\phi(r) \le \phi(s)$

$=$ $\{m \le n-1 \;\vee\; m \ge n\}$

$(m \le n-1 \;\Rightarrow\; \phi(r) \le \phi(s)) \;\wedge\; (m \ge n \Rightarrow \phi(r) \le \phi(s))$

$=$ {By definition of ϕ, $\phi(r) \le m$ and $n-1 \le \phi(s)$. Thus, $m \le n-1 \;\Rightarrow\; \phi(r) \le \phi(s)$}

$m \ge n \;\Rightarrow\; \phi(r) \le \phi(s)$

$=$ $\{m \ge n \;\equiv\; (m > n \;\vee\; m = n)\}$

$(m > n \;\Rightarrow\; \phi(r) \le \phi(s)) \;\wedge\; (m = n \;\Rightarrow\; \phi(r) \le \phi(s))$

$=$ {From previous observation, r precedes $s \;\Rightarrow\; (m > n \;\equiv\; m = n+1 \;\wedge\; i > j)$}

$((m = n+1 \;\wedge\; i > j) \;\Rightarrow\; \phi(r) \le \phi(s)) \;\wedge\; (m = n \;\Rightarrow\; \phi(r) \le \phi(s))$

$=$ {Lemma 6 in the Appendix}

$m = n \Rightarrow \phi(r) \leq \phi(s)$

$=$ {By definition of ϕ, $\phi(r) = m-1 \lor \phi(r) = m$}

$((\phi(r) = m-1 \land m = n) \Rightarrow \phi(r) \leq \phi(s)) \land$
$((\phi(r) = m \land m = n) \Rightarrow \phi(r) \leq \phi(s))$

$=$ {By definition of ϕ, $n-1 \leq \phi(s)$; thus, $m = n \Rightarrow m-1 \leq \phi(s)$}

$(\phi(r) = m \land m = n) \Rightarrow \phi(r) \leq \phi(s)$

$=$ {By definition of ϕ, $\phi(s) \leq n$}

$(\phi(r) = m \land m = n) \Rightarrow \phi(s) = n$

$=$ {By definition of ϕ and *flag*, $(\phi(r) = m) \equiv (\exists k : k < i : r!p_k)$ and
$(\phi(s) = n) \equiv (\exists k : k < j : s!p_k)$}

$((\exists k : k < i : r!p_k) \land m = n) \Rightarrow (\exists k : k < j : s!p_k)$

$=$ {$i \leq j \lor i > j$}

$((i \leq j \land (\exists k : k < i : r!p_k) \land m = n) \Rightarrow (\exists k : k < j : s!p_k)) \land$
$((i > j \land (\exists k : k < i : r!p_k) \land m = n) \Rightarrow (\exists k : k < j : s!p_k))$

$=$ {Lemmas 4 and 7 in the Appendix}

true □

6 Concluding Remarks

In order to prove the correctness of a construction, a function ϕ that meets the three conditions of safety, integrity, and precedence has to be defined for every possible history. This leads to long, and somewhat tedious proofs, mainly because we have to consider all possible ways in which Reads and Writes can overlap. To keep the resulting case analysis to a minimum, we chose the function ϕ to be very simple; it depends only on the boolean variable *flag*. If our proof appears formidable, in spite of this simplification, then it is because we have been very formal in our reasoning, so as to leave no doubt about the validity of the proof.

Our definition of atomicity is equivalent to that given by Misra in [9]. 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. There has been a lot of discussion recently on the similarities and differences between the conditions of atomicity and serializability. We refer the reader to [6] for details.

Acknowledgements: We would like to thank Ted Herman and the UT Distributed Systems

Discussion Group for their comments on this paper.

Appendix: Propositions and Lemmas

We now prove Propositions 1 through 4 and Lemmas 1 through 7. But, first we present some definitions.

Definition: Let p be an operation of program P in some history and i be a label of a statement in P. If i is not a **for** loop, then p **at** i is true at a state of the history iff the event $p\!:\!i$ is enabled; p **before** i is true at a state iff the state occurs before the event $p\!:\!i$; p **after** i is true at a state iff the state occurs after the event $p\!:\!i$.

Otherwise, if i is a **for** loop, then p **at** i is true at a state of the history iff some event $p\!:\!i.j$ of the loop is enabled; p **before** i is true at a state iff the state occurs before all events of the loop; p **after** i is true at a state iff the state occurs after all events of the loop. $\square$

The following assertions are a consequence of the preceding definition; let p and i be as given in the definition.

$$\begin{aligned}
\neg(p \textbf{ at } i) &\equiv p \textbf{ before } i \lor p \textbf{ after } i\\
\neg(p \textbf{ before } i) &\equiv p \textbf{ after } i \lor p \textbf{ at } i\\
\neg(p \textbf{ after } i) &\equiv p \textbf{ at } i \lor p \textbf{ before } i
\end{aligned}$$

Definition: Let A and B be two state assertions. The assertion A **unless** B holds iff for every pair of consecutive states in any history, if $A \land \neg B$ holds in the first state, then $A \lor B$ holds in the second state. $\square$

Definition: Consider the history $s_0 \xrightarrow{e_0} \cdots s_i \xrightarrow{e_i} s_{i+1} \cdots$. We say that s_i is the state *prior to* the event e_i and s_{i+1} is the state *following* e_i. Similarly, e_i is the event *prior to* the state s_{i+1}. $\square$

Definition: The assertion $Q(w,i,j)$, where w is a Write operation and $1 \le i < j \le m$, is defined as follows.

$$Q(w,i,j) \equiv RR[i,j].flag \land RR[i,j].seq = w!seq[i] \land RR[i,j].alt = w!alt \qquad \square$$

Proposition 1: Let r be an operation of Reader i and let w be the Write operation that determines $r!y$. Then,

$$(\exists k : k < i : r!p_k) \Rightarrow (w\!:\!1.i \prec r\!:\!1) \ .$$

Proof: Assume that $r!p_k$ holds for some $k < i$. Our proof obligation is to show that $w\!:\!1.i \prec r\!:\!1$.

By the program for the Writer, $w:1.i \prec w:3.i$. Because w determines $r!y$, $w:3.i \prec r:3$. Therefore, by transitivity, $w:1.i \prec r:3$. This implies that $w!q[i]$ is determined by either r or some predecessor of r.

We now show that r does not determine $w!q[i]$. Because $r!p_k$ holds, $r!x.seq[i] = r!y.seq[i]$. Because w determines $r!y$, $r!y.seq[i] = w!seq[i]$. Therefore, by transitivity, we have

$$r!x.seq[i] = w!seq[i] \quad . \tag{1}$$

If r determines $w!q[i]$, then $w!q[i] = r!x.seq[i]$. As w assigns $seq[i] := q[i] \oplus 1$, this implies that $w!seq[i] = r!x.seq[i] \oplus 1$, contrary to (1). Therefore, r does not determine $w!q[i]$.

Thus, we conclude that $w!q[i]$ is determined by a predecessor of r, i.e., $w:1.i \prec r:1$. □

Proposition 2: Let r be an operation of Reader i, let w be the Write operation that determines $r!y$, and let t be the state prior to the event $r:2.k$. Then, for each k, where $0 < k < i$,

$$r!p_k \;\Rightarrow\; Q(w,k,i) \text{ holds at } t.$$

Proof: Assume that $r!p_k$ holds for some k in the range $0 < k < i$. Our proof obligation is to show that $Q(w,k,i)$ holds at state t, i.e., the state prior to the event $r:2.k$.

Because $r!p_k$ is true, the following assertion holds at state t.

$$RR[k,i].flag \;\wedge\; r!x.seq[k] = RR[k,i].seq \;\wedge\; r!x.alt = RR[k,i].alt$$

Because $k > 0$, we have, by the definition of p_k, $r!x = r!y$. Because w determines $r!y$, this implies that $r!x.seq[k] = w!seq[k]$ and $r!x.alt = w!alt$. Thus, by transitivity, the following assertion holds at state t.

$$RR[k,i].flag \;\wedge\; w!seq[k] = RR[k,i].seq \;\wedge\; w!alt = RR[k,i].alt$$

Therefore, from the definition of Q, $Q(w,k,i)$ holds at state t. □

Proposition 3: Let r and s be consecutive operations by Reader i. Then,

$$s!x.seq[i] = r!x.seq[i] \;\vee\; s!x.seq[i] = r!x.seq[i] \oplus 1 \quad .$$

Proof: We prove the lemma by first showing that the following assertion is an invariant.

$$\mathrm{B} \;\equiv\; WR[i].seq[i] = RW[i] \;\vee\; WR[i].seq[i] = RW[i] \oplus 1$$

To prove that B is an invariant, we consider the assertions $B0, \ldots, B4$ defined below and show that $B0 \vee \cdots \vee B4$ is an invariant. In these assertions, we refer to the local variables $q[i]$ and $seq[i]$ of the Writer and $x.seq[i]$ of Reader i.

$$\begin{array}{lllllllllll}
\mathrm{B0} & \equiv & RW[i] & = & q[i] \oplus 1 & = & seq[i] & = & WR[i].seq[i] & = & x.seq[i] \\
\mathrm{B1} & \equiv & RW[i] & = & q[i] & = & seq[i] & = & WR[i].seq[i] & = & x.seq[i]
\end{array}$$

$$\begin{array}{lllllllllll}
\text{B2} & \equiv & RW[i]\oplus 1 & = & q[i]\oplus 1 & = & seq[i] & = & WR[i].seq[i]\oplus 1 & = & x.seq[i]\oplus 1 \\
\text{B3} & \equiv & RW[i]\oplus 1 & = & q[i]\oplus 1 & = & seq[i] & = & WR[i].seq[i] & = & x.seq[i]\oplus 1 \\
\text{B4} & \equiv & RW[i]\oplus 1 & = & q[i]\oplus 1 & = & seq[i] & = & WR[i].seq[i] & = & x.seq[i]
\end{array}$$

To see that $BO \vee \cdots \vee B4$ is an invariant, observe the following.

- The only statement that can possibly falsify B0 is the **read** by the Writer from $RW[i]$. But, executing this statement when B0 is true establishes B1.
- The only statement that can possibly falsify B1 is assignment to $seq[i]$ by the Writer. But, executing this statement when B1 is true establishes B2.
- The only statement that can possibly falsify B2 is the first **write** by the Writer to $WR[i]$. But, executing this statement when B2 is true establishes B3.
- The only statement that can possibly falsify B3 is the **read** by Reader i from $WR[i]$. But, executing this statement when B3 is true establishes B4.
- The only statement that can possibly falsify B4 is the **write** by Reader i to $RW[i]$. But, executing this statement when B4 is true establishes B0.

Thus, we conclude that $B0 \vee \cdots \vee B4$ is an invariant. This also implies that B is an invariant since $(B0 \vee \cdots \vee B4) \Rightarrow B$.

We now use B to show that the lemma holds. Our proof obligation is as follows.

$$s!x.seq[i] = r!x.seq[i] \ \vee \ s!x.seq[i] = r!x.seq[i] \oplus 1$$

Let t denote the state prior to the event $s:0$. Because r and s are consecutive, the value of $RW[i]$ at state t equals $r!x.seq[i]$, and the value of $WR[i].seq[i]$ at state t equals $s!x.seq[i]$. Since B is an invariant, either

- $WR[i].seq[i] = RW[i]$ at state t, in which case $s!x.seq[i] = r!x.seq[i]$, or
- $WR[i].seq[i] = RW[i] \oplus 1$ at state t, in which case $s!x.seq[i] = r!x.seq[i] \oplus 1$. □

Proposition 4: Let r be an operation of Reader i, and let w be a Write operation such that $r:1 \prec w:1.i$. Furthermore, let t be any state such that w **after** $1.i$ holds at t and the contents of register $RR[i,j]$ $(i < j)$ appearing at state t is written by r. Then, $Q(w,i,j)$ is false at t.

Proof: Let t be any state for which w **after** $1.i$ holds and i, j be any indices such that $i < j$. Assume that $r:1 \prec w:1.i$ and that the value of $RR[i,j]$ appearing at state t is written by r. Our proof obligation is to show that $Q(w,i,j)$ is false at t.

We first show that $r!x.seq[i]$ differs from $w!seq[i]$. Let e be the event prior to state t. Because w **after** $1.i$ holds at t, $w:1.i \preceq e$. Thus, $r:1 \prec w:1.i \preceq e$. Therefore, $w!q[i]$

is determined by either r or some successor of r, namely s. In the former case, $w!q[i] = r!x.seq[i]$. As w assigns $seq[i] := q[i] \oplus 1$, $w!seq[i] \neq r!x.seq[i]$. In the latter case, $s:1 \prec w:1.i$, and therefore, $s:1 \prec w:1.i \preceq e$. Because r writes the value appearing in $RR[i,j]$ at state t and t is the state following event e, r and s are consecutive. By Proposition 3, $w!q[i]$ equals $r!x.seq[i]$ or $r!x.seq[i] \oplus 1$. Therefore, $w!seq[i]$ equals $r!x.seq[i] \oplus 1$ or $r!x.seq[i] \oplus 2$. As $\oplus$ is modulo-3 addition, $w!seq[i] \neq r!x.seq[i]$.

Because r writes the value in $RR[i,j]$ appearing at state t,

$$RR[i,j].flag = r!flag \ \wedge \ RR[i,j].seq = r!y.seq[i] \tag{2}$$

holds at state t. Consider the two values $r!y.seq[i]$ and $w!seq[i]$. If they are equal then, because $w!seq[i] \neq r!x.seq[i]$, we have $r!x.seq[i] \neq r!y.seq[i]$ and consequently, $r!flag$ is false. Therefore, by (2), $RR[i,j].flag$ is false at t and hence, $Q(w,i,j)$ is false at t. If on the other hand, $r!y.seq[i] \neq w!seq[i]$ then by (2), $RR[i,j].seq \neq w!seq[i]$ at t and therefore, $Q(w,i,j)$ is false at t. □

Lemma 1: Let w be any Write operation and $i < j$. Then,

$$(\forall i,j :: w \ \textbf{after} \ 1.i \wedge w \ \textbf{before} \ 4 \ \Rightarrow \ \neg Q(w,i,j)) \ .$$

Proof: We prove the stronger theorem state below by induction on k. The proof of the lemma then follows from the proof of the theorem.

$$(\forall i,j,k : 0 < i < j \leq M \wedge i \leq k \leq M : w \ \textbf{after} \ 1.i \wedge w \ \textbf{before} \ 4 \ \Rightarrow \ \neg Q(w,i,j))$$

Base case: $k = 0$. Follows trivially because the range is empty.

Induction Step: $k > 0$. Let t be any state for which w **after** $1.i$ and w **before** 4 holds and i,j,k be any numbers in the required range. Our proof obligation is to show that $Q(w,i,j)$ is false at t. Consider the value of the register $RR[i,j]$ at state t. If it equals the initial value of the register then $RR[i,j].flag$ is false, and consequently, $Q(w,i,j)$ is false. Otherwise, the value in the register is written by some operation (of Reader i), namely r. Let e be the event prior to state t. Consider the events $w:1.i$ and $r:1$. Either $r:1 \prec w:1.i$ or $w:1.i \prec r:1$. If $r:1 \prec w:1.i$ then, by Proposition 4, $Q(w,i,j)$ is false at t and we are done. So, assume that $w:1.i \prec r:1$ for the remainder of the proof.

By the program for Reader i, $r:1 \prec r:3 \prec r:5.j$. Because r writes the value appearing in $RR[i,j]$ at state t, $r:5.j \preceq e$. Because w **before** 4 holds at t, $e \prec w:4.1$. Thus, by transitivity,

$$w:1.i \prec r:1 \prec r:3 \prec r:5.j \preceq e \prec w:4.1 \quad . \tag{3}$$

Therefore, $r!y$ is determined by w or the Write operation immediately preceding w. In the latter case, $r!y.alt \neq w!alt$, and therefore, $Q(w,i,j)$ is false at t. So, assume that

w determines $r!y$. By (3), $r\!:\!3 \prec w\!:\!4.i$, and therefore, $r!y.done$ is false. Hence, by the definition of p_0, $r!p_0$ is false.

Next, we use the induction hypothesis to show that $r!p_l$ is false for all l in the range $0 < l < i$. This implies that $r!flag$ is false, and therefore, $Q(w,i,j)$ is false at t, which is our proof obligation.

Let $0 < l < i$. From the program of the Writer, $w\!:\!1.l \prec w\!:\!1.i$. By assumption, $w\!:\!1.i \prec r\!:\!1$. From the program for Reader i, $r\!:\!1 \prec r\!:\!2.l \prec r\!:\!3$. Thus, by transitivity and (3),

$$w\!:\!1.l \prec w\!:\!1.i \prec r\!:\!1 \prec r\!:\!2.l \prec r\!:\!3 \prec r\!:\!5.j \preceq e \prec w\!:\!4.1 \quad . \tag{4}$$

Consider the state prior to the event $r\!:\!2.l$. From the above precedence assertion, w **before** 4 and w **after** $1.l$ at this state. By the induction hypothesis, $Q(w,l,i)$ is false at this state. Therefore,

$$\neg(\ r!v[l].flag \ \wedge\ r!v[l].seq = w!seq[l] \ \wedge\ r!v[l].alt = w!alt\ ) \quad .$$

Because, $w!seq[l] = r!y.seq[l]$ and $w!alt = r!y.alt$ (w determines $r!y$), we have

$$\neg(\ r!v[l].flag \ \wedge\ r!v[l].seq = r!y.seq[l] \ \wedge\ r!v[l].alt = r!y.alt\ ) \quad .$$

Therefore, by the definition of p_l, $r!p_l$ is false, which is our proof obligation. □

Lemma 2: Let w be any Write operation and let $i < j$. Then,

$$(\forall i, j :: w \textbf{ at } 4 \wedge Q(w,i,j) \quad \textbf{unless} \quad w \textbf{ after } 4) \ .$$

Proof: We prove the stronger theorem stated below by induction on k. The proof of the lemma then follows the proof of the theorem.

$$(\forall i, j, k : 0 < i < j \leq M \wedge i \leq k \leq M : w \textbf{ at } 4 \wedge Q(w,i,j) \quad \textbf{unless} \quad w \textbf{ after } 4)$$

Base case: $k = 0$. Follows trivially because the range is empty.

Induction Step: $k > 0$. Observe that the stated safety property is preserved trivially by each event of the Writer and all Readers different from Reader i. We show that it is also preserved by each event of Reader i. Let s be any operation of Reader i. Consider the event $s\!:\!5.j$. This is the only event that may falsify the predicate $Q(w,i,j)$. Let t be the state prior to the event and u be the state following the event. Our proof obligation is that if w **at** 4 and $Q(w,i,j)$ hold at t, then $Q(w,i,j)$ holds at u (w **at** 4 holds at u trivially). By the program for Reader i, this is equivalent to showing that if w **at** 4 and $Q(w,i,j)$ hold at t, then

$$s!flag \ \wedge\ s!y.seq[i] = w!seq[i] \ \wedge\ s!y.alt = w!alt \quad . \tag{5}$$

Because $Q(w,i,j)$ is false initially ($RR[i,j].flag$ is false), the value of $RR[i,j]$ appearing at state t is written by some operation (of Reader i), namely r. Consider the events $w\!:\!1.i$

and $r:1$. By the contrapositive of Proposition 4, $\neg(w\ \textbf{after}\ 1.i)$ holds at t or $w:1.i \prec r:1$. But, by assumption, $w\ \textbf{at}\ 4$ holds at t. Therefore, $w\ \textbf{after}\ 1.i$ holds at t. Consequently, $w:1.i \prec r:1$.

Now, we show that w determines $r!y$. By the program for Reader i, $r:1 \prec r:3 \prec r:5.j$. Because r writes the value appearing in $RR[i,j]$ at state t, $r:5.j \prec s:5.j$. Because $w\ \textbf{at}\ 4$ holds at t, $s:5.j \prec w:4.M$. Thus, by transitivity,

$$w:1.i \prec r:1 \prec r:3 \prec r:5.j \prec s:5.j \prec w:4.M \quad . \tag{6}$$

Therefore, $r!y$ is determined by w or the Write operation immediately preceding w. In the latter case, $r!y.alt \neq w!alt$, and therefore, $RR[i,j].alt \neq w!alt$ at t. Consequently, $Q(w,i,j)$ is false at t. This is contrary to our assumption. Therefore, w determines $r!y$.

From the program for the Writer, $w:0 \prec w:3.i$. Because w determines $r!y$, $w:3.i \prec r:3$. Because r precedes s, $r:3 \prec s:0 \prec s:3 \prec s:5.j$. Because $w\ \textbf{at}\ 4$ holds at t and t is the state prior to $s:5.j$, $s:5.j \prec w:4.M$. Therefore,

$$w:0 \prec w:3.i \prec r:3 \prec s:0 \prec s:3 \prec s:5.j \prec w:4.M \quad . \tag{7}$$

Therefore, w determines both $s!x$ and $s!y$. Consequently, $s!y.seq[i] = w!seq[i]$ and $s!y.alt = w!alt$. This meets two out of three of our proof obligations (5) and we are left with the proof obligation that $s!flag$ holds. This is proved next.

Because $Q(w,i,j)$ holds at t, and r writes the value appearing in register $RR[i,j]$ at state t, $r!flag$ holds, i.e., $r!p_l$ holds for some l in the range $0 \leq l < i$. We meet our remaining proof obligation by showing that $r!p_l$ implies $s!p_0 \vee s!p_l$, for every l in the range $0 \leq l < i$.

We split the proof into two parts: $l = 0$ and $0 < l < i$. If $r!p_0$ then $r!y.done$. Because w determines all the three values, $r!y$, $s!x$, and $s!y$, we have $s!x = s!y$ and $s!y.done$. Consequently, $s!p_0$ holds. This completes the first part of the proof.

For the proof of the second part, assume that $r!p_l$ holds for some l in the range $0 < l < i$. Because w determines $r!y$ and $r!p_l$ holds, by Proposition 2, $Q(w,l,i)$ is true at the state prior to the event $r:2.l$. Let us call that state t'. By the contrapositive of Lemma 1, $\neg(w\ \textbf{after}\ 1.l) \ \vee\ \neg(w\ \textbf{before}\ 4)$ holds at t'. By the definitions of **after** and **before**, $w\ \textbf{before}\ 1.l \ \vee\ w\ \textbf{at}\ 1.l \ \vee\ w\ \textbf{at}\ 4 \ \vee\ w\ \textbf{after}\ 4$ holds at t'. But, from the program for the Writer, $w:1.l \prec w:1.i$; thus, $w\ \textbf{before}\ 1.l \ \vee\ w\ \textbf{at}\ 1.l \Rightarrow w\ \textbf{before}\ 1.i$. Therefore, $w\ \textbf{before}\ 1.i \ \vee\ w\ \textbf{at}\ 4 \ \vee\ w\ \textbf{after}\ 4$ holds at t'. From (6), $w\ \textbf{before}\ 1.i$ and $w\ \textbf{after}\ 4$ is false at all states in the interval $r:1$ to $r:3$. Therefore, $w\ \textbf{at}\ 4$ holds at t'. We have shown earlier that $Q(w,l,i)$ holds at t'. Therefore, by the induction hypothesis and (7), $Q(w,l,i)$ holds for all states in the interval $r:2.l$ to $s:5.j$. In particular, $Q(w,l,i)$ holds at the state prior to the event $s:2.l$. Therefore,

$$s!v[l].flag \ \wedge\ s!v[l].seq = w!seq[l] \ \wedge\ s!v[l].alt = w!alt \quad . \tag{8}$$

Because w determines both $s!x$ and $s!y$ (shown earlier), either $s!y.done$ is true (and $s!x.seq[i] = s!y.seq[i]$) or $s!x = s!y$. In the former case, $s!p_0$ is true. In the latter case,

$$s!x.seq[l] = w!seq[l] \ \wedge\ s!x.alt = w!alt \quad .$$

Therefore, using (8) and the definition of p_l, $s!p_l$ is true. This means that in either case $s!p_0 \lor s!p_l$ holds, which is our proof obligation. $\square$

Lemma 3: Let w be any Write operation and $i < j$. Then,

$$(\forall i, j :: w \textbf{ at } 4 \land Q(w,i,j) \;\Rightarrow\; w \textbf{ after } 4.i \lor (\exists k : k < i : Q(w,k,i))) \ .$$

Proof: Assume that w **at** $4 \land Q(w,i,j) \land \neg(w$ **after** $4.i)$ holds at some state t. Our proof obligation is to show that $Q(w,k,i)$ holds at state t for some $k < i$.

Since $RR[i,j].flag$ is false in the initial state, $Q(w,i,j)$ is initially false. Therefore, assume that the value appearing in $RR[i,j]$ at state t is written by operation r (of Reader i). Because $Q(w,i,j)$ holds at state t,

$$r!flag \ \land \ r!y.seq[i] = w!seq[i] \ \land \ r!y.alt = w!alt \ . \tag{9}$$

Since $r!flag$ is true, $r!x.seq[i] = r!y.seq[i]$. Thus, by (9), the following assertion holds.

$$r!x.seq[i] = w!seq[i] \tag{10}$$

We now show that $w\!:\!1.i \prec r\!:\!1$. Assume to the contrary that $r\!:\!1 \prec w\!:\!1.i$. This implies that $w!q[i]$ is determined by either r or by a successor of r. In the former case, $w!q[i] = r!x.seq[i]$. As w assigns $seq[i] := q[i] \oplus 1$, this implies that $w!seq[i] = r!x.seq[i] \oplus 1$, which contradicts (10).

Now, consider the case in which $w!q[i]$ is determined by a successor of r. In this case, the immediate successor of r, call it s, exists. Let e be the event prior to state t. Because w **at** 4 holds at state t, $w\!:\!1.i \prec e$. Because r writes the value appearing in $RR[i,j]$ at state t, $e \prec s\!:\!5.j$. Therefore, $w\!:\!1.i \prec s\!:\!5.j$. This implies that $w!q[i]$ is not determined by a successor of s; thus, it is determined by s. By Proposition 3, $w!q[i]$ equals either $r!x.seq[i]$ or $r!x.seq[i] \oplus 1$. As w assigns $seq[i] := q[i] \oplus 1$, this implies that $w!seq[i]$ equals either $r!x.seq[i] \oplus 1$ or $r!x.seq[i] \oplus 2$. Thus, because $\oplus$ is modulo-3 addition, $w!seq[i] \neq r!x.seq[i]$, which contradicts (10). Therefore, we conclude that $w\!:\!1.i \prec r\!:\!1$.

By the program for Reader i, $r\!:\!1 \prec r\!:\!5.j$. Because r writes the value appearing in $RR[i,j]$ at state t, $r\!:\!5.j \preceq e$. Because $\neg(w$ **after** $4.i)$ holds at t, $e \prec w\!:\!4.i$. Therefore,

$$w\!:\!1.i \prec r\!:\!1 \prec r\!:\!5.j \preceq e \prec w\!:\!4.i \ . \tag{11}$$

Therefore, $r!y$ is determined by either w or the immediate predecessor of w — but, the immediate predecessor of w does not determine $r!y$ since, by (9), we have $r!y.alt = w!alt$. So, w determines $r!y$.

From (9), $r!flag$ holds. By the program for Reader i, this implies that $r!p_k$ is true for some k where $k < i$. We now show that $k > 0$. Observe that assertion (11) implies that $r\!:\!3 \prec e \prec w\!:\!4.i$. Thus, because w determines $r!y$, $r!y.done$ is false, which implies that $r!p_0$ is false. Thus, $k > 0$.

Because $k < i$, $w\!:\!1.k \prec w\!:\!1.i$. By the program for Reader i, $r\!:\!1 \prec r\!:\!2.k \prec r\!:\!5.j$. Thus, by (11), we have

$$w\!:\!1.k \prec w\!:\!1.i \prec r\!:\!1 \prec r\!:\!2.k \prec r\!:\!5.j \preceq e \prec w\!:\!4.i \quad . \tag{12}$$

Let u denote the state prior to the event $r\!:\!2.k$. Because $r!p_k$ holds ($k > 0$), by Proposition 2, $Q(w,k,i)$ holds at state u. Thus, by the contrapositive of Lemma 1, $\neg(w$ **after** $1.k) \vee \neg(w$ **before** $4)$ holds at u. But, by assertion (12), w **after** $1.k$ holds at u. Consequently, $\neg(w$ **before** $4)$ holds at u, i.e, w **at** 4 or w **after** 4 holds at u. But, by (12), w **after** 4 does not hold at u. Thus, w **at** 4 holds at u, which implies that $w\!:\!3.1 \prec r\!:\!2.k$. Thus, by assertion (12), we have,

$$w\!:\!3.1 \prec r\!:\!2.k \prec r\!:\!5.j \preceq e \prec w\!:\!4.i \quad .$$

Observe that w **at** 4 holds for all states between $r\!:\!2.k$ and $w\!:\!4.i$. Thus, by Lemma 2, $Q(w,k,i)$ holds for all states in this interval. In particular, it holds at state t (the state following event e), which establishes our proof obligation. □

Lemma 4: Let r be any operation of Reader i and s be any operation of Reader j such that $i \leq j$ and r precedes s. Assume that both $r!y$ and $s!y$ are determined by the same Write operation. Then,

$$(\exists k : k < i : r\!:\!p_k) \;\Rightarrow\; (\exists l : l < j : s\!:\!p_l) \;.$$

Proof: Assume that Write operation w determines both $r!y$ and $s!y$ and that $r!p_k$ holds for some $k < i$. Our proof obligation is to show that $s!p_l$ for some $l < j$.

We first establish that w determines both $s!x$ and $s!y$. Because $i \leq j$, we have $w\!:\!3.j \preceq w\!:\!3.i$. Because w determines $r!y$, $w\!:\!3.i \prec r\!:\!3$. Because r precedes s, $r\!:\!3 \prec s\!:\!0$. By the program for Reader j, $s\!:\!0 \prec s\!:\!3$. Therefore, by transitivity, we get,

$$w\!:\!3.j \preceq w\!:\!3.i \prec r\!:\!3 \prec s\!:\!0 \prec s\!:\!3 \quad . \tag{13}$$

Because w determines $s!y$, by (13), w determines both $s!x$ and $s!y$.

Now, consider the events $w\!:\!4.j$ and $s\!:\!3$. Either $w\!:\!4.j \prec s\!:\!3$ or $s\!:\!3 \prec w\!:\!4.j$. We first dispose of the former case by showing that $s!p_0$ is true. Because w determines both $s!x$ and $s!y$, $s!x.seq[j] = s!y.seq[j]$. Moreover, since $w\!:\!4.j \prec s\!:\!3$, $s!y.done$ is true. Therefore, by the definition of p_0, $s!p_0$ is true. In the remainder of the proof, we assume that $s\!:\!3 \prec w\!:\!4.j$. By transitivity on (13), we have,

$$w\!:\!3.j \preceq w\!:\!3.i \prec r\!:\!3 \prec s\!:\!0 \prec s\!:\!3 \prec w\!:\!4.j \quad . \tag{14}$$

By assumption, $r!p_k$ holds for some $k < i$. Thus, by Proposition 1, $w\!:\!1.i \prec r\!:\!1$. From the program for Reader i, $r\!:\!1 \prec r\!:\!3$. Therefore, by (14),

$$w\!:\!1.i \prec r\!:\!1 \prec r\!:\!3 \prec s\!:\!0 \prec s\!:\!3 \prec w\!:\!4.j \quad . \tag{15}$$

Next, we show that $Q(w, l, j)$ is true at the state prior to $s\!:\!2.l$ for some l in the range $0 < l < j$. As a result, the following assertion holds.

$$s!v[l].flag \;\wedge\; s!v[l].seq = w!seq[l] \;\wedge\; s!v[l].alt = w!alt$$

Observe that, by (14), $s!x = s!y$, $s!x.seq[l] = w!seq[l]$, and $s!x.alt = w!alt$. Therefore,

$$s!x = s!y \;\wedge\; s!v[l].flag \;\wedge\; s!v[l].seq = s!x.seq[l] \;\wedge\; s!v[l].alt = s!x.alt \quad .$$

By the definition of p_l, $s!p_l$ is true, which is our proof obligation.

We show that $Q(w, l, j)$ is true, for some l, in the state prior to $s\!:\!2.l$, by considering the two possiblities $i = j$ and $i < j$ separately.

$\underline{i = j}$: In this case, we show that $l := k$, i.e., $Q(w, k, j)$ is true prior to the event $s\!:\!2.k$. By (14), $r!y.done$ is false, which implies that $r!p_0$ is also false. Because $r!p_k$ is true for some k, we have $k > 0$. Let t be the state prior to $r\!:\!2.k$. By Proposition 2, $Q(w, k, i)$ holds at t. Thus, by the contrapositive of Lemma 1, $\neg(w$ **after** $1.k) \vee \neg(w$ **before** $4)$ holds at t.

Because $k < i$, we have $w\!:\!1.k \prec w\!:\!1.i$. By the program for Reader i, $r\!:\!1 \prec r\!:\!2.k \prec r\!:\!3$. Thus, by (15),

$$w\!:\!1.k \prec w\!:\!1.i \prec r\!:\!1 \prec r\!:\!2.k \prec r\!:\!3 \prec s\!:\!0 \prec s\!:\!3 \prec w\!:\!4.i \quad .$$

Therefore, w **after** $1.k$ holds at state t. Consequently, $\neg(w$ **before** $4)$ holds at t, i.e., w **at** 4 or w **after** 4 holds at t. But, by the above precedence assertion, w **after** 4 does not hold at t. Thus, w **at** 4 holds at t, i.e., $w\!:\!3.1 \prec r\!:\!2.k$. Therefore,

$$w\!:\!3.1 \prec r\!:\!2.k \prec s\!:\!0 \prec s\!:\!3 \prec w\!:\!4.i \quad .$$

Observe that w **at** 4 holds for all states between $r\!:\!2.k$ and $s\!:\!3$. Thus, by Lemma 2, $Q(w, k, i)$ holds for all states in that interval. In particular, it holds in the state prior to $s\!:\!2.k$.

$\underline{i < j}$: In this case, we show that $l := i$, i.e., $Q(w, i, j)$ is true prior to the event $s\!:\!2.i$. Let t be the state following the event $r\!:\!5.j$. Because $r!p_k$ holds, $r!flag$ is true. Therefore, the following assertion holds at state t.

$$RR[i,j].flag \;\wedge\; RR[i,j].seq = r!y.seq[i] \;\wedge\; RR[i,j].alt = r!y.alt$$

Because w determines $r!y$, $r!y.seq[i] = w!seq[i]$ and $r!y.alt = w!alt$. Thus, the following assertion holds at state t.

$$RR[i,j].flag \;\wedge\; RR[i,j].seq = w!seq[i] \;\wedge\; RR[i,j].alt = w!alt$$

Therefore, $Q(w, i, j)$ is true at state t. Thus, by the contrapositive of Lemma 1, $\neg(w$ **after** $1.i)$ $\vee\ \neg(w$ **before** $4)$ holds at state t.

By the program for Reader i, we have $r\!:\!3 \prec r\!:\!5.j$. Since r precedes s, we have $r\!:\!5.j \prec s\!:\!0$. Thus, by (15),

$$w\!:\!1.i \prec r\!:\!1 \prec r\!:\!3 \prec r\!:\!5.j \prec s\!:\!0 \prec s\!:\!3 \prec w\!:\!4.j \quad .$$

Therefore, w **after** $1.i$ holds at state t. Consequently, $\neg(w$ **before** $4)$ holds at t, i.e., w **at** 4 or w **after** 4 holds at t. But, by the above precedence assertion w **after** 4 does not hold at t. Thus, w **at** 4 holds at t, i.e., $w\!:\!3.1 \prec r\!:\!5.j$. Therefore,

$$w\!:\!3.1 \prec r\!:\!5.j \prec s\!:\!0 \prec s\!:\!3 \prec w\!:\!4.j$$

Observe that w **at** 4 holds for all states between $r\!:\!5.j$ and $s\!:\!3$. Thus, by Lemma 2, $Q(w,i,j)$ holds for all states in that interval. In particular, it holds in the state prior to $s\!:\!2.i$. $\Box$

Lemma 5: Let w be any Write operation and $i < j$. Then,

$$(\forall i,j :: w \textbf{ at } 4 \wedge Q(w,i,j) \;\Rightarrow\; (\forall k : i < k \leq j : Q(w,i,k))) \; .$$

Proof: Assume that w **at** 4 $\wedge$ $Q(w,i,j)$ holds at some state t. Let $i < k < j$. We show that $Q(w,i,k)$ holds at state t.

Because $RR[i,j].flag$ is false in the initial state, $Q(w,i,j)$ is initially false. Therefore, assume that the value appearing in $RR[i,j]$ at state t is written by operation r (of Reader i).

Observe that if r writes the value appearing in $RR[i,k]$ at state t, then because $Q(w,i,j)$ holds at t, $Q(w,i,k)$ also holds at t. Thus, in the remainder of the proof assume that a succeeding operation (of Reader i), call it s, writes the value appearing in $RR[i,k]$ at state t.

Let e denote the event prior to state t. Because $k < j$, $r\!:\!5.k \prec r\!:\!5.j$. Because r precedes s, $r\!:\!5.j \prec s\!:\!5.k$. Because s writes the value appearing in $RR[i,k]$ at state t, $s\!:\!5.k \preceq e$. Because r writes the value appearing in $RR[i,j]$ at state t, $e \prec s\!:\!5.j$. Therefore,

$$r\!:\!5.k \prec r\!:\!5.j \prec s\!:\!5.k \preceq e \prec s\!:\!5.j \quad . \tag{16}$$

Because r writes the value appearing in $RR[i,j]$ at state t, (16) implies that s is the immediate successor of r. Because $Q(w,i,j)$ holds at t,

$$r!flag \;\wedge\; r!y.seq[i] = w!seq[i] \;\wedge\; r!y.alt = w!alt \quad . \tag{17}$$

Since $r!flag$ holds, $r!x.seq[i] = r!y.seq[i]$. Thus, we have

$$r!x.seq[i] = w!seq[i] \quad . \tag{18}$$

We now show that $w\!:\!1.i \prec r\!:\!1$. Assume to the contrary that $r\!:\!1 \prec w\!:\!1.i$. Because w **at** 4 holds at state t, $w\!:\!1.i \prec e$. Thus, by (16), we have $r\!:\!1 \prec w\!:\!1.i \prec s\!:\!5.j$. This implies that $w!q[i]$ is determined by r or by s. Thus, by Proposition 3, $w!q[i]$ equals either $r!x.seq[i]$ or $r!x.seq[i] \oplus 1$. As w assigns $seq[i] := q[i] \oplus 1$, this implies that $w!seq[i]$ equals either $r!x.seq[i] \oplus 1$ or $r!x.seq[i] \oplus 2$. Thus, because $\oplus$ is modulo-3 addition, $w!seq[i] \neq r!x.seq[i]$, which contradicts (18). Therefore, we conclude that $w\!:\!1.i \prec r\!:\!1$.

By the program for Reader i, $r\!:\!1 \prec r\!:\!5.k$. Since w **at** 4 holds at t, $e \prec w\!:\!4.M$. Thus, by (16), we have

$$w\!:\!1.i \prec r\!:\!1 \prec r\!:\!5.k \prec r\!:\!5.j \prec s\!:\!5.k \preceq e \prec w\!:\!4.M \quad . \tag{19}$$

Thus, $r!y$ is determined by either w or the immediate predecessor of w — but, the immediate predecessor of w does not determine $r!y$ since, by (17), we have $r!y.alt = w!alt$. Therefore, w determines $r!y$. This implies that $w\!:\!3.i \prec r\!:\!3$. By the program for Reader i, $r\!:\!3 \prec r\!:\!5.k$. Hence, by (19), we have

$$w\!:\!3.i \prec r\!:\!3 \prec r\!:\!5.k \prec r\!:\!5.j \prec s\!:\!5.k \preceq e \prec w\!:\!4.M \quad .$$

Because r precedes s, this precedence assertion implies that w determines $s!y$.

Because $Q(w, i, j)$ holds at state t, $r!flag$ is true. Furthermore, as established above, w determines both $r!y$ and $s!y$. Hence, by Lemma 4, $s!flag$ is true. Therefore, because w determines $s!y$, we have the following assertion.

$$s!flag \ \wedge \ s!y.seq[i] = w!seq[i] \ \wedge \ s!y.alt = w!alt$$

Therefore, because s writes the value appearing in $RR[i, k]$ at state t, by the definition of Q, $Q(w, i, k)$ is true at state t. □

Lemma 6: Let r be any operation of Reader i and s be any operation of Reader j such that $i > j$ and r precedes s. Assume that $r!y$ and $s!y$ are determined by Write operations $W\!:\!m$ and $W\!:\!n$ respectively where $m = n + 1$. Then, $\phi(r) \leq \phi(s)$.

Proof: We meet the proof obligation, $\phi(r) \leq \phi(s)$, by showing that $\phi(r) = \phi(s) = m-1$.

Let w denote $W\!:\!n$ and w' denote $W\!:\!m$. (Because $n < m$, w precedes w'.) Because w' determines $r!y$, $w'\!:\!3.i \prec r\!:\!3$. Because r precedes s, $r\!:\!3 \prec s\!:\!0$. By the program for Reader j, $s\!:\!0 \prec s\!:\!3$. Because w determines $s!y$ and w precedes w', $s\!:\!3 \prec w'\!:\!3.j$. Therefore, by transitivity,

$$w'\!:\!3.i \prec r\!:\!3 \prec s\!:\!0 \prec s\!:\!3 \prec w'\!:\!3.j \quad . \tag{20}$$

Hence, $r!y.done$ is false, $s!y.done$ is true, and $s!x.seq[j] = s!y.seq[j]$. Thus, by the definition of p_0, $r!p_0$ is false and $s!p_0$ is true. Because $s!p_0$ is true, from the definition of ϕ, $\phi(s) = n = m-1$. In the remainder of the proof, we show that $\phi(r) = m-1$. Since $r!p_0$ is false, by the definition of ϕ, our proof obligation is to show that $r!p_k$ is false for each k in the range $0 < k < i$.

Because w' determines $r!y$, $w':1.i \prec r:3$. Consider the two events $r:1$ and $w':1.i$. Either $r:1 \prec w':1.i$ or $w':1.i \prec r:1$. In the former case, by the contrapositive of Proposition 1, $r!p_k$ is false for each k, which is our proof obligation.

We now show that $r!p_k$ is false for each k, where $0 < k < i$, for the other alternative, i.e., $w':1.i \prec r:1$. Because $k < i$, $w':1.k \prec w':1.i$. By the program for Reader i, $r:1 \prec r:2.k \prec r:3$. Thus, using (20), we have

$$w':1.k \prec w':1.i \prec r:1 \prec r:2.k \prec r:3 \prec s:0 \prec s:3 \prec w':3.j \quad .$$

Note that w' **after** $1.k \wedge w'$ **before** 4 holds at the state prior to $r:2.k$. From Lemma 1, $\neg Q(w', k, i)$ holds at that state. Therefore, by the contrapositive of Proposition 2, $r!p_k$ is false, which is the required proof obligation. □

Lemma 7: Let r be any operation of Reader i and s be any operation of Reader j such that $i > j$ and r precedes s. Assume that both $r!y$ and $s!y$ are determined by the same Write operation. Then,

$$(\exists k : k < i : r{:}p_k) \;\Rightarrow\; (\exists l : l < j : s{:}p_l) \; .$$

Proof: Assume that Write operation w determines both $r!y$ and $s!y$ and that $r!p_k$ holds for some $k < i$. Our proof obligation is to show that $s!p_l$ for some $l < j$.

Consider the two events $w:4.i$ and $r:3$. Either $w:4.i \prec r:3$ or $r:3 \prec w:4.i$. We first dispose of the former case. Because $j < i$, we have $w:4.j \prec w:4.i$. Because r precedes s, $r:3 \prec s:0$. By the program for Reader j, $s:0 \prec s:3$. Therefore,

$$w:4.j \prec w:4.i \prec r:3 \prec s:0 \prec s:3 \quad .$$

Thus, since w determines $s!y$, $s!y.done$ is true and $s!x.seq[j] = s!y.seq[j]$. Therefore, by the definition of p_0, $s!p_0$ is true, which establishes our proof obligation.

In the remainder of the proof, assume that $r:3 \prec w:4.i$. Because $r!p_k$ holds, by Proposition 1, $w:1.i \prec r:1$. By the program for Reader i, $r:1 \prec r:2.k \prec r:3$. Therefore,

$$w:1.i \prec r:1 \prec r:2.k \prec r:3 \prec w:4.i \quad .$$

Let t denote the state prior to the event $r:2.k$. We now show that w **at** 4 holds at t. Because $r!p_k$ is true, by Proposition 2, $Q(w, k, i)$ holds at state t. By the contrapositive of Lemma 1, $\neg(w$ **after** $1.k) \;\vee\; \neg(w$ **before** $4)$ holds at t. Because, $k < i$, we have $w:1.k \prec w:1.i$. Thus, by the previous precedence assertion, we have the following.

$$w:1.k \prec w:1.i \prec r:1 \prec r:2.k \prec r:3 \prec w:4.i$$

Therefore, w **after** $1.k$ holds at state t. Consequently, $\neg(w$ **before** $4)$ holds at t, i.e, w **at** 4 or w **after** 4 holds at t. But, by the above precedence assertion, w **after** 4 does not hold at t. Thus, w **at** 4 holds at t.

Because $i > j \geq 1$, we have $w\!:\!3.i \prec w\!:\!3.j \preceq w\!:\!3.1$. Since w **at** 4 holds at state t, we have $w\!:\!3.1 \prec r\!:\!2.k$. By the program for Reader i, $r\!:\!2.k \prec r\!:\!3$. Therefore,

$$w\!:\!3.i \prec w\!:\!3.j \preceq w\!:\!3.1 \prec r\!:\!2.k \prec r\!:\!3 \quad . \tag{21}$$

Consider the two events $w\!:\!4.j$ and $s\!:\!3$. Either $w\!:\!4.j \prec s\!:\!3$ or $s\!:\!3 \prec w\!:\!4.j$. We first dispose of the former case. Because r precedes s, by (21), we have $w\!:\!3.j \prec s\!:\!0$. Since w determines $s!y$ and $w\!:\!4.j \prec s\!:\!3$, $s!y.done$ is true. Moreover, since $w\!:\!3.j \prec s\!:\!0$, $s!x.seq[j] = s!y.seq[j]$. Therefore, by the definition of p_0, $s!p_0$ is true, which establishes our proof obligation.

In the remainder of the proof, assume that $s\!:\!3 \prec w\!:\!4.j$. Because r precedes s, $r\!:\!3 \prec s\!:\!0 \prec s\!:\!3$. Thus, by (21), the following assertion holds.

$$w\!:\!3.i \prec w\!:\!3.j \preceq w\!:\!3.1 \prec r\!:\!2.k \prec r\!:\!3 \prec s\!:\!0 \prec s\!:\!3 \prec w\!:\!4.j \tag{22}$$

Therefore, $s!x = s!y$.

As we show below, there exists i', where $i' < j$, such that $Q(w, i', j)$ holds at state t (the state prior to the event $r\!:\!2.k$). By assertion (22), w **at** 4 holds for all states between $r\!:\!2.k$ and $s\!:\!3$. Thus, by Lemma 2, $Q(w, i', j)$ holds for all states in this interval. In particular, it holds in the state following the execution of the event $s\!:\!2.i'$. Therefore,

$$s!v[i'].flag \ \wedge \ s!v[i'].seq = w!seq[i'] \ \wedge \ s!v[i'].alt = w!alt \quad .$$

Because w determines $s!y$, $s!y.seq[i'] = w!seq[i']$ and $s!y.alt = w!alt$. Since $s!x = s!y$, $s!x.seq[i'] = s!y.seq[i']$ and $s!x.alt = s!y.alt$. Therefore,

$$s!x = s!y \ \wedge \ s!v[i'].flag \ \wedge \ s!x.seq[i'] = s!v[i'].seq \ \wedge \ s!x.alt = s!v[i'].alt \quad .$$

Consequently, by the definition of $p_{i'}$, $s!p_{i'}$ is true, which is our proof obligation.

We now prove that the i' mentioned above exists; this is our last remaining proof obligation. As established above, w **at** $4 \wedge Q(w, k, i)$ holds at state t. Thus, by repeated application of Lemma 3, there exists a strictly decreasing sequence of Reader indices $\sigma_0, \ldots, \sigma_L$ such that:

- $\sigma_0 = i$, and $\sigma_1 = k$.
- For each m', where $0 \leq m' < L$, $Q(w, \sigma_{m'+1}, \sigma_{m'})$ holds at state t.
- w **after** $4.\sigma_L$ holds at state t.

The existence of this sequence depends upon the fact that the set of pairs of Reader indices is well-founded; thus, Lemma 3 cannot be repeatedly applied "forever."

By the program for the Writer, $w\!:\!3.j \prec w\!:\!4.\sigma_L$. Since w **after** $4.\sigma_L$ holds at t, we have $w\!:\!4.\sigma_L \prec r\!:\!2.k$. Thus, by (22), we have

$$w\!:\!3.j \prec w\!:\!4.\sigma_L \prec r\!:\!2.k \prec r\!:\!3 \prec s\!:\!0 \prec s\!:\!3 \prec w\!:\!4.j \quad .$$

By the above precedence assertion, $w:4.\sigma_L \prec w:4.j$; this implies that $\sigma_L < j$. Therefore, because $j < i$ and $i = \sigma_0$, we have $\sigma_L < j < \sigma_0$. Thus, there exists n' such that $\sigma_{n'+1} < j \leq \sigma_{n'}$.

Observe that w **at** $4 \ \wedge \ Q(w, \sigma_{n'+1}, \sigma_{n'})$ holds at state t. Thus, by Lemma 5, $(\forall m' : \sigma_{n'+1} < m' \leq \sigma_{n'} : Q(w, \sigma_{n'+1}, m'))$ also holds at state t. Hence, because $\sigma_{n'+1} < j \leq \sigma_{n'}$, $Q(w, \sigma_{n'+1}, j)$ holds at state t. Thus, we take $i' := \sigma_{n'+1}$; this establishes our final proof obligation. □

References

[1] J. Anderson and M. Gouda, The virtue of patience: concurrent programming with and without waiting, unpublished manuscript.

[2] J. Anderson, A. Singh, and M. Gouda, The elusive atomic register, *Technical Report TR.86.29*, Department of Computer Sciences, University of Texas at Austin, 1986.

[3] B. Bloom, Constructing two-writer atomic registers, *IEEE Transactions on Computers*, Vol. 37, No. 12, December 1988, pp. 1506-1514. Also appeared in *Proceedings of the Sixth Annual Symposium on Principles of Distributed Computing*, 1987, pp. 249-259.

[4] J. Burns and G. Peterson, Constructing multi-reader atomic values from non-atomic values, *Proceedings of the Sixth Annual Symposium on Principles of Distributed Computing*, 1987, pp. 222-231.

[5] P. Courtois, F. Heymans, and D. Parnas, Concurrent control with readers and writers, *Communications of the ACM*, Vol. 14, No. 10, Oct. 1971, pp. 667-668.

[6] Herlihy, M. and J. Wing, Axioms for Concurrent Objects, Technical Report CMU-CS-86-154, Computer Science Department, Carnegie Mellon University, 1986.

[7] A. Israeli and M. Li, Bounded time-stamps, *Proc. of 28th IEEE Symposium on Foundations of Computer Science*, 1987, pp. 371- 382.

[8] L. Lamport, On interprocess communication, parts I and II, *Distributed Computing*, Vol. 1, pp. 77-101, 1986.

[9] J. Misra, Axioms for memory access in asynchronous hardware systems, *ACM Transactions on Programming Languages and Systems*, Vol. 8, pp 142-153, 1986.

[10] R. Newman-Wolfe, A protocol for wait-free, atomic, multi-reader shared variables, *Proceedings of the Sixth Annual Symposium on Principles of Distributed Computing*, 1987, pp. 232-248.

[11] G. Peterson, Concurrent reading while writing, *ACM Transactions on Programming Languages and Systems*, Vol. 5, pp. 46-55, 1983.

[12] G. Peterson and J. Burns, Concurrent reading while writing II: the multi-writer case, *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, 1987.

[13] A. Singh, J. Anderson, and M. Gouda, The elusive atomic register, revisited, *Proceedings of the Sixth Annual Symposium on Principles of Distributed Computing*, 1987, pp. 206-221.

[14] P. Vitanyi and B. Awerbuch, Atomic shared register access by asynchronous hardware, *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, 1986, pp. 233-243.