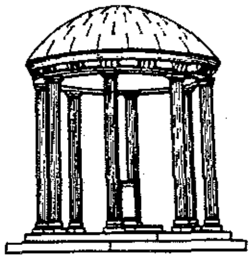
Upper and Lower Bounds for One-Write Multivalued Regular Registers

TR91-026

June, 1991

Soma Chaudhuri
Martha J. Kosa
Jennifer L. Welch

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175



UNC is an Equal Opportunity/Affirmative Action Institution.

Upper and Lower Bounds for One-Write Multivalued Regular Registers

Soma Chaudhuri Martha J. Kosa[†] Jennifer L. Welch

Department of Computer Science

University of North Carolina at Chapel Hill

Campus Box 3175, Sitterson Hall

Chapel Hill, NC 27599-3175

June 10, 1991

Abstract

This paper presents an algorithm for implementing a k-ary regular register (the *logical* register) using $k(k-1)/2$ binary regular registers (the *physical* registers) that requires only one physical write per logical write. The algorithm is simple to describe and depends on properties of paths in a related graph. Two lower bounds on the number of registers required by one-write implementations are given. The first lower bound holds for a restricted class of implementations and implies that our algorithm is optimal for this class. The second lower bound, $2k - 1 - \lceil \log k \rceil$, holds for a more general class of algorithms. Both lower bounds improve on the best previously known lower bound, which was k. Both lower bounds use a general technique that we have formalized for "fooling the reader" into violating the regular property. Our second lower bound uses a general result for transforming a one-write algorithm into another one-write algorithm for fewer logical values using fewer physical registers. We show that for any one-write algorithm, there is no advantage, in terms of number of physical registers, to be gained if readers write, or if different readers follow different protocols, or if a reader's protocol depends on its history. Furthermore, for our second class of algorithms, there is also no advantage to be gained if the reader reads some physical registers more than once. These results are shown using transformation techniques similar to the one mentioned previously.

[†]Contact Author. E-mail: kosa@cs.unc.edu

1 Introduction

In any concurrent system, processes need to communicate with other processes. Concurrent reads and writes of shared memory cells, or registers, are required for communication. If the shared memory provides more guarantees, then it is more useful to the users of the system, but implementing the shared memory may be more difficult. Thus it is helpful to know which types of registers can implement which other types and what the costs of these implementations are. Many such implementations have been developed [Blo87, BP87, Lam86, LTV90, NW87, Pet83, SAG87, Tro89, VA86, Vid88, CW90].

In this paper we focus on implementing a k-ary regular register, the *logical* register, out of binary regular registers, the *physical* registers, for $k > 2$. A *register* is a memory cell that supports concurrent reading and writing by a collection of processes; we assume there are several readers but only one writer. A *k-ary* register can take on k different values; *binary* means 2-ary. The term "regular" refers to the consistency guarantee provided in the presence of overlapping reads and writes: a read of a *regular* register must return either the value of the most recent preceding write (a well-defined notion since there is only one writer) or the value of an overlapping write. Although regularity is not as strong a guarantee as *atomicity*, which means that the values returned by reads are consistent with some total ordering on all the operations that respects the relative orderings of the operations, it seems to be much cheaper to implement and may very well be sufficient for many applications. These definitions were introduced by Lamport [Lam86].

More specifically, we are interested in *one-write algorithms*—implementations with the property that every WRITE to the logical register requires only one write to a physical register*. Since bounds on the number of physical accesses per logical access can be converted into time bounds for the logical access, a one-write algorithm would have time-efficient logical WRITEs, perhaps an important characteristic for applications in which WRITEs outnumber READs.

In this paper, we present a one-write algorithm for implementing a k-ary regular register out of binary regular registers. Clearly this algorithm is optimal in the number of physical writes per logical WRITE. The best previous upper bound was $\lceil \log k \rceil$ writes per WRITE, due to Chaudhuri and Welch [CW90]. The algorithm is simple to describe using the complete graph whose nodes are labeled with the logical values. Its correctness proof is based on properties of paths in this graph.

One drawback of our algorithm is that it requires $k(k-1)/2 = O(k^2)$ physical registers. The best previous lower bound on the number of physical registers for a k-ary implementation was k [CW90], for any number of physical writes per logical WRITE. Thus binary to k-ary implementations are inherently expensive in the amount of "hardware" required. In this paper we show two improved lower bounds on the number of physical registers in any one-write algorithm. Each lower bound holds for a natural class of implementations. The first lower bound holds for a restricted class of implementations satisfying the *toggle* property. This lower bound implies that our algorithm is optimal in the number of physical registers for this class. The second

*The names of logical operations will be capitalized in the remainder of this paper, and the names of physical operations will remain in lower case.

lower bound, $2k-1-\lfloor \log k \rfloor$, holds for a more general and reasonably unrestrictive class of implementations satisfying the *symmetric* property.

Our lower bounds are proved by contradiction; in both cases, the ultimate contradiction reached is a violation of the regular property. We have formalized a general technique for proving that the regular property does not hold by "fooling the reader" into returning an incorrect value. We also developed a general transformation to convert a one-write algorithm for k values into a one-write algorithm for $k-1$ values using fewer physical registers. This transformation is used in the inductive proof of our symmetric lower bound. In proving these two lower bounds, we have developed considerable understanding of such one-write algorithms. We can prove that, for any one-write algorithm, there is no advantage, in terms of number of physical registers, to be gained if readers write, or if different readers follow different protocols, or if a reader's protocol depends on its history. Furthermore, for symmetric algorithms, there is no advantage if a reader reads some physical registers more than once. Thus our lower bound proofs are simpler, since we assume the reader does none of the above. These results are shown with transformation techniques similar to the one mentioned previously.

In Section 2, we present our basic definitions. Section 3 contains the algorithm and in Section 4 we prove it is correct. Section 5 consists of our lower bounds. We conclude in Section 6.

Some of the results of this paper have appeared in preliminary form in [CKW91].

2 Definitions

We model each system component with an automaton. The automaton is a state machine whose state transitions are labeled with **actions**. An **execution** of an automaton is an alternating sequence of states and actions, beginning with an initial state, in which each action is enabled in the previous state and each state change correctly reflects the transition relation for the intervening action.

The complete system is also modeled with an automaton, the composition of the components. Components communicate by sharing actions with the same name. When a shared action occurs, all components sharing that action change state simultaneously.

Given a value set V and initial value $v_0 \in V$, a **logical register implementation** is the composition of n readers, one writer, and m physical registers, defined below.

The i^{th} **reader**, $1 \le i \le n$, is an automaton that satisfies the following.

- It interacts with the environment of the logical register using actions READ(i) and RETURN(i, v), $v \in V$.
- It reads some subset of the physical registers using actions $\text{read}_j(i)$ and $\text{return}_j(i, v)$, $v \in \{0, 1\}$, where j ranges over the physical registers read.

- It writes some (possibly empty) subset of the physical registers using actions $\text{write}_j(i, v)$, $v \in \{0, 1\}$, and ack_j, where j ranges over the physical registers written.

The **writer** is an automaton that satisfies the following.

- It interacts with the environment of the logical register using actions WRITE(v), $v \in V$, and ACK.
- It reads some (possibly empty) subset of the physical registers using actions $\text{read}_j(0)$ and $\text{return}_j(0, v)$, $v \in \{0, 1\}$, where j ranges over the physical registers read. (The first argument of 0 indicates the writer.)
- It writes some subset of the physical registers using actions $\text{write}_j(v)$, $v \in \{0, 1\}$, and ack_j, where j ranges over the physical registers written.

Given the value set $\{0, 1\}$ and initial value w_j, **physical register** X_j, $1 \leq j \leq m$, is an automaton that satisfies the following.

- It interacts with some subset of the read and write processes using actions $\text{read}_j(i)$ and $\text{return}_j(i, v)$, $v \in \{0, 1\}$, where i ranges over the processes in the subset.
- It interacts with exactly one of the readers and the writer using $\text{write}_j(v)$ and ack_j actions, $v \in \{0, 1\}$.

In the complete system, the physical actions of the physical registers and of the readers and the writer must "match up", i.e., X_j has physical action ϕ if and only if exactly one reader or writer has physical action ϕ.

Each reader and the environment cooperate so that, for all i, READ(i) and RETURN($i, *$) actions alternate, starting with a READ(i). A READ(i) and its following RETURN($i, *$) form a **logical READ operation**. A logical READ is **pending** if it lacks its RETURN. Similarly, the writer and the environment cooperate so that WRITE and ACK actions alternate, starting with a WRITE. A WRITE and its following ACK form a **logical WRITE operation**. A logical WRITE is **pending** if it lacks its ACK.

Each reader, as well as the writer, cooperates with each physical register so that, for all i and j, $\text{read}_j(i)$ and $\text{return}_j(i, *)$ actions alternate, starting with a $\text{read}_j(i)$. A $\text{read}_j(i)$ and its following $\text{return}_j(i, *)$ form a **physical read operation**. A physical read is **pending** if it lacks its return. Similarly, each reader, as well as the writer, cooperates with each physical register so that, for all i and j, $\text{write}_j(i, *)$ and $\text{ack}_j(i)$ actions alternate, starting with a $\text{write}_j(i, *)$. A $\text{write}_j(i, *)$ and its following $\text{ack}_j(i)$ form a **physical write operation**. A physical write is **pending** if it lacks its ack.

We assume that the physical registers are regular and wait-free, i.e., each X_j satisfies the following two conditions.

- (Physical regular property) In every execution, every read_j returns the value of an overlapping write_j or the value of the most recent preceding write_j (the initial value w_j if there is no preceding write_j).
- (Physical wait-free property) For every finite execution in which X_j has a pending physical operation, the action to complete the operation is enabled in the last state of the execution.

In the complete system, the readers and the writer must ensure that the logical register is regular and wait-free, i.e., that the following two conditions hold.

- (Logical regular property) In every execution, every READ RETURNs the value of an overlapping WRITE or the value of the most recent preceding WRITE.
- (Logical wait-free property) For every finite execution in which a reader or the writer has a pending logical operation, there is a finite extension in which no other reader or writer takes steps and the operation completes.

For simplicity in our proofs, we assume that each reader and the writer is quiescent (does not access the physical registers) unless a logical operation is pending at that reader (or writer).

To describe a register implementation algorithm, it is sufficient to describe the code for the readers and the writer. An algorithm is a **one-write algorithm** if, in every execution, every logical WRITE uses at most one physical write.

We now define several terms which will be used in the discussion of one-write algorithms.

Let A be a one-write algorithm that uses m binary registers to implement a k-ary register with value set V and initial value v_0. A **configuration** of A is an element C of $\{0,1\}^m$; let $C[i]$ denote the i^{th} bit of C for $i \in \{1,\ldots,m\}$. The **distance** between two configurations C_1 and C_2, denoted $d(C_1, C_2)$, is the number of bits that differ in C_1 and C_2. Configurations C_1 and C_2 are **neighbors** if $d(C_1, C_2) = 1$. A configuration C is **initial** if $C[i]$ is the value of the i^{th} binary register in the initial state of A for all $i \in \{1,\ldots,m\}$. A configuration C is **reachable** if there exists a state in an execution of A where no physical write is pending such that $C[i]$ is the value of the i^{th} binary register in the state for all $i \in \{1,\ldots,m\}$. (If a physical write is pending, the value of that physical register is ambiguous.)

3 The Algorithm

In this section, we present our one-write algorithm.

Let V be the value set of the logical register, where $|V| = k$ and $v_0 \in V$ is the initial value. Let K_V be the complete graph with k nodes and $r = C(k,2)$ edges in which each edge is labeled with a distinct number from the set $\{1,\ldots,r\}$ and each node is labeled with a distinct element from V. The **special bit**

set corresponding to $v \in V$ is defined as $s(v) = \{l \in \{1, \ldots, r\} : l$ labels an edge incident to the node labeled v in $K_V\}$. Since K_V is a complete graph, $|s(v)| = k - 1$ for all $v \in V$.

Our algorithm uses r binary regular registers (bits). Each bit corresponds to an edge of K_V. A reader reads all r bits and returns the value of a function f applied to the configuration obtained. The function f is defined below. The writer changes a bit only when the value of the logical register changes; when the value is changed from v to w, the bit whose label is contained in $s(v) \cap s(w)$ is changed. There is exactly one such bit because there is exactly one edge connecting v and w in K_V. Figure 1 is a formal description of our algorithm.

We now define f, the **value extraction function**. For each $v \in V$ and configuration C, let $count(C, v) = |\{i \in s(v) : C[i] = 1\}|$. Configuration C is **valid** if either (1) $count(C, v)$ is even for all $v \in V$, or (2) $count(C, v_0)$ is odd and $count(C, w)$ is odd for exactly one $w \neq v_0$. Otherwise, C is invalid. First we define f for valid configuration C. If $count(C, v)$ is even for all $v \in V$, then let $f(C) = v_0$. Otherwise, let $f(C) = v$, where $v \neq v_0$ and $count(C, v)$ is odd. Now we define f for invalid configurations. Let c be the **closest valid configuration function**, where $c(C)$ is defined to be the first configuration in lexicographic order in the set $\{D : D$ is valid and $d(C, D)$ is a minimum$\}$. Define $f(C)$, for C not valid, to be $f(D)$, where $D = c(C)$.

If a configuration C is valid, then there is a path in K_V, not necessarily edge-disjoint, starting from the node labeled with v_0 and initial configuration 0^r such that when the path is traversed and the appropriate bits are changed, then the resulting configuration is C. The resulting node is labeled v, where $v = f(C)$. For each $i \in \{1, \ldots, r\}$, $C[i]$ is the parity of the number of times edge i is traversed in this path. We now explain the graph-theoretic concepts supporting our definition of count. Suppose the path corresponding to valid configuration C is noncyclic. The two endpoints of the path are adjacent to an odd number of edges in the path, while all internal nodes are adjacent to an even number. The last node in the path is entered one more time than it is left; thus, the count for that node is odd. The first node in the path is left one more time than it is entered; thus, the count for that node is odd. All other nodes are entered and left the same number of times; thus, the counts for those nodes are even. C satisfies condition (2) of the definition of valid. Suppose the path corresponding to valid configuration C is cyclic. All nodes in the cycle are adjacent to an even number of edges in the cycle. All nodes in the cycle are entered and left the same number of times; thus, the counts for all the nodes are even. C satisfies condition (1) of the definition of valid.

4 Proof of Correctness

In this section, we prove that our algorithm implements a k-ary regular register from binary regular registers. The bulk of this section is devoted to showing that the logical register satisfies the regular property.

Lemma 4.1 shows that any reachable configuration is valid and is mapped by f to the value which was written to the register by the last completed WRITE.

```
Physical Registers (Bits): $X_1, \ldots, X_r$, initially $X_j = 0$, for all $j \in \{1, \ldots, r\}$
Reader $i$, $1 \le i \le n$: variables $x_1, \ldots, x_r$
    READ($i$):
        for $j := 1$ to $r$ do
            read$_j(i)$
            return$_j(i, x_j)$
        endfor
    RETURN($i, f(x_0 \ldots x_{r-1})$)

Writer: variables $x_1, \ldots, x_r$, initially $x_j = 0$, for all $j \in \{1, \ldots, r\}$, and
                        $old$, initially $old = v_0$
    WRITE($v$):
        if $v \neq old$ then
            pick $i$ from $s(v) \cap s(old)$
            write$_i(\overline{x_i})$
            ack$_i$
            $x_i := \overline{x_i}$
            $old := v$
        endif
    ACK
```

Figure 1: One-Write Algorithm

Lemma 4.1 *Let C be a reachable configuration resulting from a sequence of m physical writes corresponding to the logical values $v_1, v_2, \ldots, v_m$. Then C is valid, and $f(C) = v_m$.*

Proof We proceed by induction on m.

Basis: ($m = 0$.) Then C is the initial configuration and is valid, and $f(C) = v_0$.

Inductive step: ($m > 0$.) Suppose the lemma is true for $m-1$. Now we show that it is true for m. Suppose the sequence of logical values is $v_1, v_2, \ldots, v_{m-1}, v_m$ and the sequence of corresponding reachable configurations is $C_1, C_2, \ldots, C_{m-1}, C_m$. By the inductive hypothesis, C_{m-1} is valid, and $f(C_{m-1}) = v_{m-1}$. If $v_{m-1} = v_m$, then C_m trivially is valid, and $f(C_{m-1}) = f(C_m)$ because $C_m = C_{m-1}$. Thus, suppose that $v_{m-1} \neq v_m$. There are two possibilities for v_{m-1}. Either $v_{m-1} = v_0$, or $v_{m-1} \neq v_0$.

Case 1: $v_{m-1} = v_0$. Then $count(C_{m-1}, v)$ is even for all $v \in V$. When the write for v_m is performed, the unique bit $b \in s(v_0) \cap s(v_m)$ is changed. Thus $count(C_m, v_0)$ and $count(C_m, v_m)$ become odd, and $count(C_m, v)$ remains even for all $v \in V - \{v_0, v_m\}$. Therefore C_m is valid, and $f(C_m) = v_m$.

Case 2: $v_{m-1} \neq v_0$. Then $count(C_{m-1}, v_0)$ and $count(C_{m-1}, v_{m-1})$ are odd, and $count(C_{m-1}, v)$ is even for all $v \in V - \{v_0, v_{m-1}\}$. When the write for v_m is performed, the unique bit $b \in s(v_{m-1}) \cap s(v_m)$ is changed. There are two possibilities for v_m. Either $v_m = v_0$, or $v_m \neq v_0$. First suppose that $v_m = v_0$. Thus $count(C_m, v_0)$ and $count(C_m, v_{m-1})$ become even, and $count(C_m, v)$ remains even for all $v \in V - \{v_0, v_{m-1}\}$. Therefore C_m is valid, and $f(C_m) = v_0$. Now suppose that $v_m \neq v_0$. Thus $count(C_m, v_m)$ becomes odd, $count(C_m, v_0)$ remains odd, and $count(C_m, v)$ is even for all $v \in V - \{v_0, v_m\}$. Therefore C_m is valid, and $f(C_m) = v_m$. ■

We need to show that the logical register implemented by our algorithm satisfies the regular property. If a reader RETURNs value v, we must show that v was actually written to the register by some WRITE overlapping the READ or by the last WRITE preceding the READ. This is nontrivial because a slow reader can read either a reachable or a nonreachable configuration by noticing traces from many WRITEs to the logical register by a fast writer. Lemma 4.2 shows that a WRITE(v) operation has occurred during an interval in an execution if a bit in $s(v)$ is changed during that interval. Lemma 4.3 shows that if two valid configurations agree in all bits of $s(v)$ for some v and one is mapped to v by the value extraction function, then the other must be mapped to v by the value extraction function. Lemma 4.4 shows that an invalid configuration C agrees with its closest valid configuration C_N in the special bits corresponding to $f(C_N)$. Lemma 4.5, which shows that the reader will RETURN a correct value of the register no matter what configuration it reads, is the main result of this section. The proof of Lemma 4.5 uses Lemma 4.2 initially to deduce that if a value is not written to the logical register, then its special bit set remains unchanged. If the reader reads a reachable configuration, then Lemma 4.3 is applied to deduce the correctness of the value RETURNed. Otherwise, Lemmas 4.4 and 4.3 are applied to deduce the correctness of the value RETURNed.

Lemma 4.2 *For any interval in any execution, if no WRITE(v) operation overlaps the interval or occurs as the last preceding WRITE, then the bits in $s(v)$ are not changed during the interval.*

Proof Suppose in contradiction that a bit in $s(v)$ is changed during the interval. Then the value in the register changed from some w to v or the value in the register changed from v to some w. This is impossible because no WRITE(v) operation overlapped the interval or occurred as the last preceding WRITE. Therefore, the lemma is true. ■

Lemma 4.3 *Choose any valid configurations C and D. If $f(D) = v$ and $C[i] = D[i]$ for all $i \in s(v)$, then $f(C) = v$.*

Proof There are two cases to consider. Either $v = v_0$, or $v \neq v_0$.

Case 1: $v = v_0$. Thus $count(D, w)$ is even for all $w \in V$. Since $C[i] = D[i]$ for all $i \in s(v_0)$, $count(C, v_0) = count(D, v_0)$. Thus $count(C, w)$ is even for all $w \in V$ because C is valid. This implies that $f(C) = v_0$.

Case 2: $v \neq v_0$. Thus $count(D, v)$ is odd. Since $C[i] = D[i]$ for all $i \in s(v)$, $count(C, v) = count(D, v)$. Thus $count(C, v_0)$ is odd and $count(C, w)$ is even for all $w \in V - \{v_0, v\}$ because C is valid. This implies that $f(C) = v$. ■

Lemma 4.4 *Choose any invalid configuration C. Let $D = c(C)$ and $v = f(D)$. Then $C[i] = D[i]$ for all $i \in s(v)$.*

Proof Suppose in contradiction that there exists at least one bit $b \in s(v)$ such that C and D are not equal in that bit. Thus $d(C, D) = l \geq 1$. Change bit b in D to yield C_D. C_D is valid and $C_D[b] = C[b]$. So $d(C, C_D) = l - 1$. This is a contradiction, because D was supposed to be the closest valid configuration to C. Therefore, the lemma is true. ■

Lemma 4.5 *Let C be the configuration obtained by a reader during some execution of the READ protocol. Suppose $f(C) = v$. Then the value v was written by a WRITE which overlapped the READ or the value v was the result of the last WRITE preceding the READ.*

Proof Assume for contradiction that the value v was not written by a WRITE which overlapped the READ and the value v was not the result of the last WRITE preceding the READ. Thus no state of the algorithm during the READ has the physical registers in a configuration with value v. By Lemma 4.2, the bits in $s(v)$ are never changed during the READ. Let D be any reachable configuration resulting from either the last preceding WRITE or any overlapping WRITE. D is valid by Lemma 4.1, and $D[i] = C[i]$ for all $i \in s(v)$. There are two cases to consider. Either C is a valid configuration, or C is an invalid configuration.

Case 1: Suppose C is valid. Since D has the same values as C for the bits in $s(v)$ and $f(C) = v$, $f(D) = v$ by Lemma 4.3, which is a contradiction.

Case 2: Suppose C is not valid. Let $C_N = c(C)$. Then $f(C_N) = v$. By Lemma 4.4, $C[i] = C_N[i]$ for all $i \in s(v)$. By the transitive property of equality, $C_N[i] = D[i]$ for all $i \in s(v)$. By Lemma 4.3, $f(D) = v$, which is a contradiction. ■

The result of Lemma 4.5 proves the following theorem. The logical register is seen to be wait-free by inspecting the code of the read and write processes.

Theorem 4.1 *A one-write algorithm for implementing a k-ary regular register from binary regular registers exists.*

5 Lower Bounds on Number of Registers

We have proven the existence of a one-write algorithm for implementing a k-ary regular register from binary regular registers. The number of registers used by our algorithm is very large, $C(k, 2) = O(k^2)$. The best previously known lower bound on the number of registers for this problem is k, shown by Chaudhuri and Welch [CW90]. In this section we establish lower bounds on the number of registers required by two classes of one-write algorithms. Subsection 5.1 gives a lower bound of $C(k, 2)$ for the class of one-write algorithms satisfying the toggle property. Subsection 5.2 gives a lower bound of $2k-1-\lfloor \log k \rfloor$ for the class of one-write algorithms satisfying the symmetric property. These properties are defined below.

A one-write algorithm with the following properties is a **normal form** algorithm.

1. no reader performs a physical write
2. every reader has the same program
3. every reader starts in the same state at the beginning of every READ

In Subsection 5.3, we show how any one-write algorithm can be converted to a normal form algorithm without increasing the number of physical registers. Thus we can, without loss of generality, restrict our attention to normal form algorithms.

If one-write algorithm A uses m binary registers, A has 2^m configurations. These configurations are nodes in a directed m-dimensional hypercube H_A. If configurations C_1 and C_2 are neighbors, then both (C_1, C_2) and (C_2, C_1) are edges of H_A. An edge (C_1, C_2) of H_A is an **algorithm edge** if C_1 and C_2 are reachable configurations and C_2 can be derived from C_1 after one WRITE operation. An edge (C_1, C_2) of H_A is labeled with i, where i is the bit in which C_1 and C_2 differ.

A one-write algorithm A has the **symmetric property** if for all configurations C_1, C_2 that are neighbors, (C_1, C_2) is an algorithm edge of H_A if and only if (C_2, C_1) is an algorithm edge of H_A. If A satisfies the symmetric property, the two directed edges connecting any pair of neighboring configurations are either both algorithm edges or both non-algorithm edges. Thus the two directed edges can be replaced by one edge which is either an algorithm edge or a non-algorithm edge. Therefore, H_A can be considered an undirected graph. In Subsection 5.3, we show how an arbitrary symmetric algorithm can be transformed into a symmetric algorithm using no more registers in which every reader reads each physical register at most once during a READ. Thus we can assume without loss of generality that in a symmetric algorithm every reader reads each physical register at most once during a READ. The symmetric property seems reasonably unrestrictive and it may allow for implementations requiring fewer physical registers.

A one-write algorithm has the **toggle property** if for each pair of distinct $v, w \in V$, there exists a bit l such that whenever the value of the logical register is changed from v to w or from w to v, bit l is written. A one-write algorithm satisfying the toggle property trivially satisfies the symmetric property. The toggle property is an overly restrictive property for a one-write algorithm. Our algorithm satisfies this property. We will show that our algorithm is optimal in the class of algorithms satisfying this property with respect to the number of physical registers.

In our lower bound proofs, we want to deduce the value which must be RETURNed by a reader given a particular configuration of the physical registers. This is analogous to the value extraction function from our algorithm in Section 3. However, for our algorithm, every reader reads every physical register exactly once during a READ. This makes f easily defined but overly restrictive for proofs of lower bounds. We consider a more general class of symmetric algorithms in which a reader does not have to read all the physical registers. Thus we need a slightly more complex definition of a general value extraction function. We first define the term consistent. Bit i is **consistent** with configuration C if the value of bit i is $C[i]$. Let A be a symmetric algorithm for implementing a k-ary regular register from m binary regular registers. For algorithm A we define a **general value extraction function** $f_A : \{0,1\}^m \rightarrow V$. If no reader ever reads bits consistent with configuration C, then $f_A(C)$ is undefined. If all the bits that a reader reads are consistent with configuration C and the reader RETURNs v, then $f_A(C) = v$. Thus f_A is a partial function. We now discuss why f_A is well-defined. Consider two logical READs. Suppose the reader performing the first logical READ reads a subset S_1 of the physical registers, RETURNing v_1, and the reader performing the second logical READ reads a different subset S_2 of the physical registers, RETURNing v_2, where $v_1 \neq v_2$. Suppose all bits in $S_1 \cup S_2$ are consistent with C. This is impossible because the readers have the same program and start their READs in the same initial state. For the readers to read two different sets of physical registers, there must be some physical register for which the first reader obtained 1 and the second reader obtained 0 (or vice versa). Thus one of the readers did not read bits consistent with configuration C. Therefore, f_A is well-defined.

We now define terms which will be used in the formalization of our general technique for "fooling the reader", which is Lemma 5.1. Let A be a one-write algorithm for implementing a k-ary regular register from m binary regular registers that satisfies the symmetric property. Let S be a set of reachable configurations and C be a configuration. C is **constructible** from S for each $i \in \{1, \ldots, m\}$, there exists a $C' \in S$ such that $C'[i] = C[i]$. (A similar definition was given in [JSL90].) Let $f_A(S) = \{f_A(C) : C \in S\}$. S is **connected** if for all distinct $D, E \in S$, there exists a path from D to E in H_A consisting only of algorithm edges in which every configuration on the path is an element of S.

Given a configuration C which is constructible from a connected set of configurations S, Lemma 5.1 states that $f_A(C)$ must be in $f_A(S)$. In our lower bound proofs, we try to contradict Lemma 5.1 (build an execution in which a reader is "fooled") in order to obtain the desired lower bounds. We obtain a contradiction by identifying a connected set S of configurations and showing how there is a constructible C with the wrong value. We call this our "fooling the reader" technique.

Lemma 5.1 *Let A be a one-write algorithm that satisfies the symmetric property. For all configurations C and connected sets of reachable configurations S, if C is constructible from S, then $f_A(C) \in f_A(S)$.*

Proof Suppose in contradiction that $f_A(C) \notin f_A(S)$. Consider the following execution of A. First the writer executes a sequence of WRITEs so that the resulting configuration of the physical registers is in S. This sequence exists because S is reachable. Then a logical READ starts. For all i, whenever the reader is about to read bit i, the writer executes a sequence of WRITEs with the following properties: (1) the configuration of the physical registers after each WRITE is in S, and (2) the final configuration D is such that $C[i] = D[i]$. Since S is connected, this sequence exists. Thus the reader returns $f_A(C)$, which violates the regular property because $f_A(C)$ was not the value of any overlapping WRITE or of the preceding WRITE. ■

This lemma is true for nonsymmetric algorithms if S is a strongly connected set and the definition of f_A is appropriately modified. In the general case, we can define $f_A(C)$ to be the value RETURNed by a reader if all the bits that a reader reads are consistent with configuration C and if the reader never sees two different values for the same bit during the READ. The lemma might be useful in proving lower bounds for nonsymmetric algorithms.

5.1 Toggle Property

We can show that the upper bound of $C(k,2)$ is tight for the class of algorithms satisfying the toggle property (which includes our algorithm). Every algorithm A with the toggle property can be represented by the complete graph on k nodes, in which each node is labeled with a distinct element from V and the edge between v and w is labeled with some $l \in \{1,\ldots,m\}$ (when the value of the logical register is changed from v to w or vice versa, bit l is changed), where m is the number of binary registers used by A. Call this graph G_A.

When $k=3$, $k=C(k,2)$; thus our algorithm is trivially optimal in the number of binary regular registers used. Theorem 5.1 shows that $C(k,2)$ binary regular registers are necessary for any $k \geq 4$.

Theorem 5.1 *For all one-write algorithms A for implementing a k-ary ($k \geq 4$) regular register from binary regular registers, if A has the toggle property, then the number of binary regular registers used by A is at least $C(k,2)$.*

Proof Suppose that A is a one-write algorithm for implementing a k-ary regular register from binary regular registers, where A has the toggle property and the number of registers used by A is less than $C(k,2)$. Then there is some register i such that i is the label of at least two edges in G_A, say (v_1,v_2) and (v_3,v_4). Suppose the edges have a common endpoint. Without loss of generality, assume $v_1=v_3$. Then $v_2 \neq v_4$ because otherwise the edges would be the same. If the current value of the logical register is v_1 and bit i is changed, the new value of the logical register is both v_2 and v_4, which is ambiguous. Thus the edges are disjoint; v_1, v_2, v_3, and v_4 are distinct.

Let j, where $j \neq i$, label the edge (v_1,v_3) of G_A. Let C_1 be any configuration such that $f_A(C_1)=v_1$. Let C_2 be the configuration that differs from C_1 only in bit i. Let C_3 be the configuration that differs from C_1 only in bit j. Let C_4 be the configuration that differs from C_1 only in bits i and j. By the definition of G_A, C_2, C_3, and C_4 are reachable configurations, and $f_A(C_2)=v_2$, $f_A(C_3)=v_3$, and $f_A(C_4)=v_4$. Figure 2 shows the relationships among C_1,C_2,C_3, and C_4. C_2 is constructible from the connected set $\{C_1,C_3,C_4\}$. But $f_A(C_2)=v_2$ is not in $f_A(\{C_1,C_3,C_4\})=\{v_1,v_3,v_4\}$, contradicting Lemma 5.1. ∎

5.2 Symmetric Property

The symmetric property seems to be desirable since it seems intuitive that an algorithm with this property would use fewer registers. Also, it makes a lower bound proof easier since we can use the "fooling the

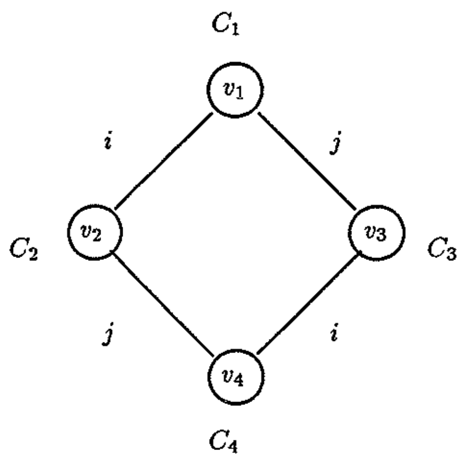


Figure 2: Relationships Among the Four Configurations in the Proof of Theorem 5.1

reader" technique. Let $SYM(k)$ be the set of all one-write algorithms which implement a k-ary regular register from binary regular registers and satisfy the symmetric property. For an algorithm $A \in SYM(k)$, let $R_A(k)$ be the number of binary registers used by A. Let $R(k)$ be the minimum number of binary registers required by any one-write algorithm in $SYM(k)$. The main result of this section is Theorem 5.2, which states that $R(k) > 2k - 2 - \lfloor \log k \rfloor$. The proof of Theorem 5.2 is inductive. Lemma 5.2, which shows that 4 binary regular registers cannot implement a 4-ary regular register, forms the base case for the proof. In the inductive step, either k is a power of 2, or k is not a power of 2. If k is a power of 2, then Lemma 5.4, which proves that $R(k) \geq R(k-1) + 1$, is used. If k is not a power of 2, then Lemma 5.5, which proves that $R(k) \geq R(k-1) + 2$, is used. Then some algebraic manipulations enable us to derive the desired lower bound. The proofs of Lemmas 5.4 and 5.5 use Lemma 5.3, which gives conditions under which a one-write algorithm can be converted into a one-write algorithm for fewer logical values using fewer physical registers. The proof of Lemma 5.3 consists of a general algorithm transformation.

Lemma 5.2 $R(4) > 4$.

Proof Suppose in contradiction that there exists an algorithm A such that $R_A(4) = 4$. Suppose without loss of generality that $V = \{R, G, B, Y\}$, the initial configuration is 0000, $f_A(0000) = R$, $f_A(1000) = G$, $f_A(0100) = B$, and $f_A(0010) = Y$. We now attempt to assign values to the remaining 12 configurations.

Figure 3 shows the current assignment of values to configurations and the possibilities for some currently unassigned configurations. Because A is a one-write algorithm, we only need to consider configurations which differ in one bit from the last assigned configuration 1000. We cannot assign two different values to the same configuration. Thus, we have six choices to consider:

1. $f_A(1010) = B$ and $f_A(1100) = Y$.
2. $f_A(1010) = B$ and $f_A(1001) = Y$.
3. $f_A(1001) = B$ and $f_A(1100) = Y$.
4. $f_A(1100) = B$ and $f_A(1010) = Y$.
5. $f_A(1100) = B$ and $f_A(1001) = Y$.
6. $f_A(1001) = B$ and $f_A(1010) = Y$.

We can eliminate choices 1 and 2 by showing that $f_A(1010) \neq B$. $f_A(1010) \neq B$ because otherwise 0010 is constructible from the connected set $\{0000, 1000, 1010\}$ and $f_A(0010) = Y$ is not in $f_A(\{0000, 1000, 1010\}) = \{R, G, B\}$, contradicting Lemma 5.1. We can eliminate choice 3 by showing that $f_A(1100) \neq Y$. $f_A(1100) \neq Y$ because otherwise 0100 is constructible from the connected set $\{0000, 1000, 1100\}$ and $f_A(0100) = B$ is not in $f_A(\{0000, 1000, 1100\}) = \{R, G, Y\}$, contradicting Lemma 5.1.

We now show how to eliminate choices 4, 5, and 6. We consider each of the three choices in turn.

Case 4. Figure 4 shows the current assignment of values to configurations and the possibilities for some currently unassigned configurations. $f_A(0110) \neq G$ because otherwise 0110 is constructible from the connected set $\{0000, 0010, 0100\}$ and $f_A(0110)$ is not in $f_A(\{0000, 0010, 0100\}) = \{R, B, Y\}$, contradicting Lemma 5.1. Thus, we only have one choice to consider: $f_A(0101) = G$ and $f_A(0110) = Y$. Figure 5 shows the current assignment of values to configurations and the possibilities for some currently unassigned configurations. f_A cannot map 0011 to both G and B. This case leads to a dead end.

Case 5. Figure 6 shows the current assignment of values to configurations and the possibilities for some currently unassigned configurations. As in Choice 4, $f_A(0110) \neq G$ and thus $f_A(0101) = G$ and $f_A(0110) = Y$. Figure 7 shows the current assignment of values to configurations and the possibilities for some currently unassigned configurations. $f_A(1010) \neq B$ because otherwise 1000 is constructible from the connected set $\{0000, 0010, 1010\}$ and $f_A(1000) = G$ is not in $f_A(\{0000, 0010, 1010\}) = \{R, B, Y\}$, contradicting Lemma 5.1. Thus, we only have one choice to consider: $f_A(1010) = G$ and $f_A(0011) = B$. Figure 8 shows the current assignment of values to configurations and the possibilities for some currently unassigned configurations. $f_A(1101) \neq R$ because otherwise 1001 is constructible from the connected set $\{0000, 1000, 1100, 1101\}$ and $f_A(1001) = Y$ is not in $f_A(\{0000, 1000, 1100, 1101\}) = \{R, G, B\}$, contradicting Lemma 5.1. $f_A(1110) \neq R$ because otherwise 0110 is constructible from the connected set $\{0000, 1000, 1100, 1110\}$ and $f(0110) = Y$ is not in $f_A(\{0000, 1000, 1100, 1110\}) = \{R, G, B\}$, contradicting Lemma 5.1. This case leads to a dead end.

Case 6. Figure 9 shows the current assignment of values to configurations and the possibilities for some currently unassigned configurations. $f_A(0110) \neq G$ because otherwise 1010 is constructible from the connected set $\{0000, 1000, 0100, 0110\}$ and $f_A(1010) = Y$ is not in $f_A(\{0000, 1000, 0100, 0110\}) = \{R, G, B\}$, contradicting Lemma 5.1. $f_A(1100) \neq Y$ because otherwise 1000 is constructible from the connected set

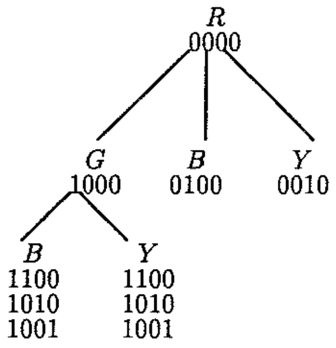


Figure 3: First Set of Choices

$\{0000, 0100, 1100\}$ and $f_A(1000) = G$ is not in $f_A(\{0000, 0100, 1100\}) = \{R, B, Y\}$, contradicting Lemma 5.1. Thus, we have three choices to consider:

6.1. $f_A(1100) = G$ and $f_A(0110) = Y$.

6.2. $f_A(1100) = G$ and $f_A(0101) = Y$.

6.3. $f_A(0101) = G$ and $f_A(0110) = Y$.

We consider each of the three choices in turn.

Case 6.1. Figure 10 shows the current assignment of values to configurations and the possibilities for some currently unassigned configurations. f_A cannot map 0011 to both G and B. This choice leads to a dead end.

Case 6.2. Figure 11 shows the current assignment of values to configurations and the possibilities for some currently unassigned configurations. $f_A(0110) \neq G$ because otherwise 0100 is constructible from the connected set $\{0000, 0010, 0110\}$ and $f_A(0100) = B$ is not in $f_A(\{0000, 0010, 0110\}) = \{R, G, Y\}$, contradicting Lemma 5.1. $f_A(0011) \neq G$ because otherwise 1001 is constructible from the connected set $\{0000, 1000, 0010, 0011\}$ and $f_A(1001) = B$ is not in $f_A(\{0000, 1000, 0010, 0011\}) = \{R, G, Y\}$, contradicting Lemma 5.1. This case leads to a dead end.

Case 6.3. Figure 12 shows the current assignment of values to configurations and the possibilities for some currently unassigned configurations. f_A cannot map 0011 to both G and B. We have nowhere else to backtrack.

Thus, $R(4) > 4$.

■

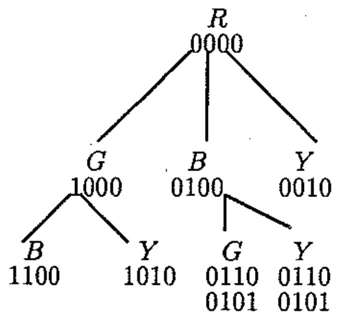


Figure 4: Case 4 and Second Set of Choices

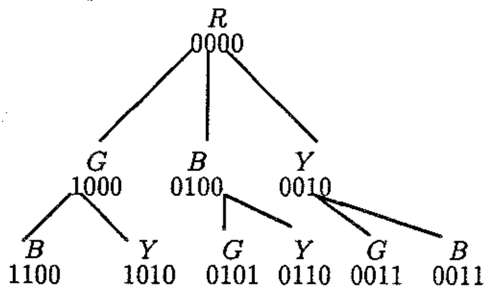


Figure 5: Case 4 - Remaining Choice

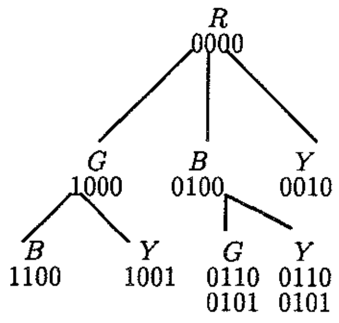


Figure 6: Case 5 and Second Set of Choices

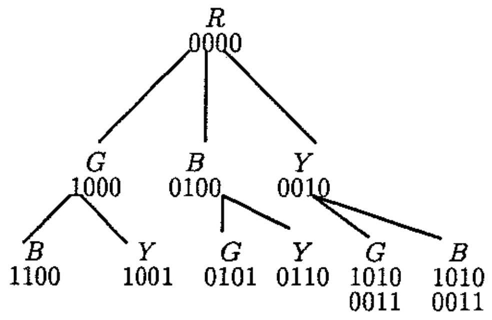


Figure 7: Case 5 and Third Set of Choices

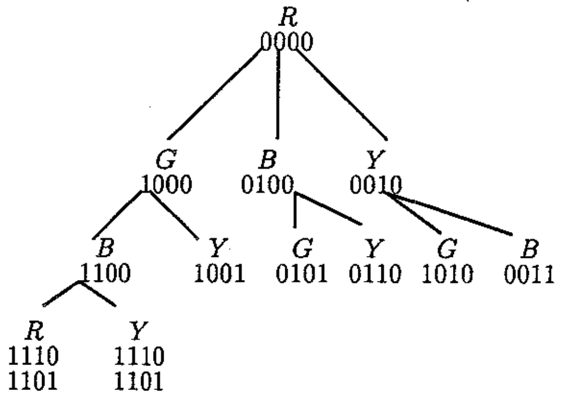


Figure 8: Case 5 and Fourth Set of Choices

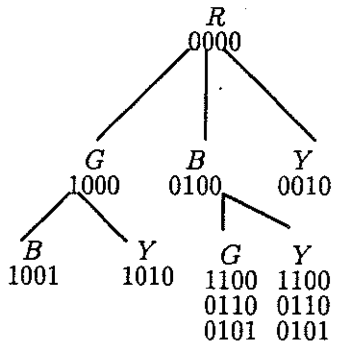


Figure 9: Case 6 and Second Set of Choices

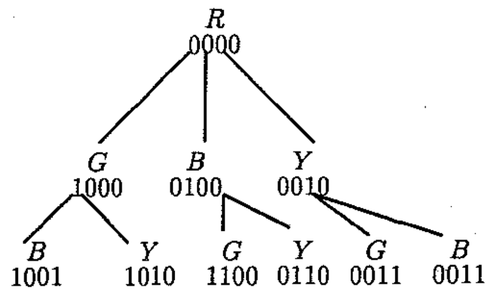


Figure 10: Case 6.1

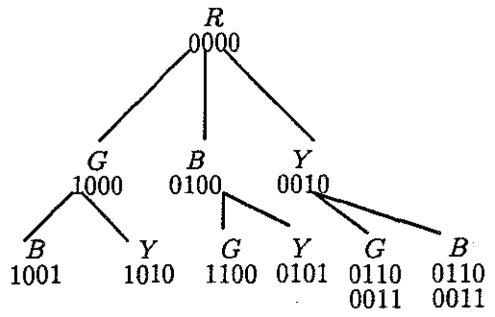


Figure 11: Case 6.2

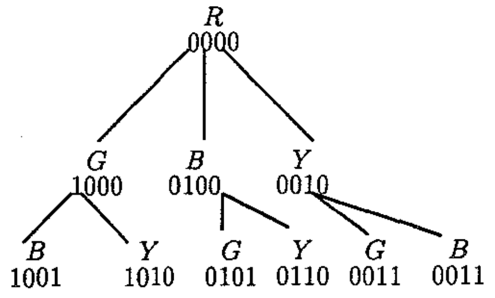


Figure 12: Case 6.3

Lemma 5.3 *Consider any $A \in SYM(k)$ with $R_A(k) = m$. Suppose there exists a reachable configuration C and a value $w \neq f_A(C)$ such that C has p neighbors D with $f_A(D) = w$. Then there exists a one-write algorithm $A' \in SYM(k-1)$ with $R_{A'}(k-1) \leq m - p$.*

Proof We show how to construct A' given A. A' will implement a logical register with value set $V - \{w\}$, where V is the value set of the logical register implemented by A, and initial value $v_0 \in V - \{w\}$.

For each $i \in \{1, \ldots, p\}$, let C_i be the neighbor of C that differs from C in bit b_i, where $f_A(C_i) = w$. Consider the set S of all configurations L reachable from C by a path of algorithm edges in which no configuration X with $f_A(X) = w$ appears in the path. Let Z be the subgraph of H_A in which the node set is S and the edge set is the set of all edges in $S \times S$ that are algorithm edges in H_A. No edge in Z is labeled with any bit in $\{b_1, b_2, \ldots, b_p\}$ because otherwise some C_i is constructible from S, which is connected, and $f_A(C_i) = w$ is not in $f_A(S)$, contradicting Lemma 5.1.

Algorithm A' will use $m - p$ binary regular registers. We now define the initial configuration for A'. Assume without loss of generality that b_1 through b_p are the last p bits and they are all 0 in C. Thus, b_1 through b_p are all 0 in every configuration in S. Given $D \in S$, define $\pi(D)$ to be the prefix of D consisting of all but the last p bits. (These will be the reachable configurations of A'.) If $f_A(C) = v_0$, let $D_0 = C$. Otherwise, let D_0 be the neighbor of C in Z such that $f_A(D_0) = v_0$. Clearly D_0 exists. We define the initial configuration of A' to be $\pi(D_0)$.

We now describe the reader's protocol in algorithm A'. The reader's protocol in algorithm A' is the same as the reader's protocol in algorithm A, except that the reader in A' has local bits $c_1, \ldots, c_p$ corresponding to shared bits $b_1, \ldots, b_p$ in A. The value of bit c_i is 0 for each $i \in \{1, \ldots, p\}$ at all times. Whenever reader j in A reads shared bit b_i, the reader in A' reads local bit c_i using action localread(j, c_i).

We now describe the writer's protocol in algorithm A'. If the current configuration of the physical registers (well-defined because readers do not write) is $\pi(E)$ for some $E \in S$ and WRITE(x), for x not the current value of the logical register, is the next operation, then the writer changes bit b, where b labels the algorithm edge (E, D) in Z and $f_A(D) = x$. An easy induction shows that in every state of every execution of A' the physical registers always form a configuration $\overline{E}$ such that $\overline{E} = \pi(E)$ for some $E \in S$.

Now we must show that algorithm A' implements a $(k-1)$-ary regular register. Algorithm A' clearly holds $(k-1)$ values and satisfies the wait-free property. We now show that the regular property holds. Consider any execution e' of algorithm A'. We build a corresponding execution e of algorithm A as follows. We construct a sequence of actions of A by starting with a sequence of logical WRITEs to ensure that the configuration of the physical registers is D_0. We then consider each action in the execution of A' in turn. If the action is not a read of a local bit c_i by reader j, then the action is placed as is in the sequence. If the action is a read of a local bit c_i by reader j, then the actions read$_{b_i}(j)$ and return$_{b_i}(j, 0)$ are placed in order in the sequence. By construction, there exists an execution e of A with the sequence of actions just constructed. By the assumption about A, e satisfies the regular property. Suppose a READ by reader j in execution e' of algorithm A' RETURNs value v. Then the corresponding READ in the constructed execution e of algorithm A also RETURNs value v. We must prove that v is a proper value to RETURN in e'. In e, v is the value of an overlapping WRITE, the value of the last preceding WRITE, or the initial value of A. We consider each possibility in turn. If in e, v is the value of an overlapping WRITE, then WRITE(v) overlaps the original READ in e'. Thus v is a proper value to RETURN in e'. If in e, v is the value of the last preceding WRITE, then either there is a corresponding WRITE(v) in e' or there is not a corresponding WRITE(v) in e' (so no WRITE precedes the READ in e'). If there is a corresponding WRITE(v) in e', then v is a proper value to RETURN in e'. Otherwise v is v_0, the initial value for A'; thus v is a proper value to RETURN in e'. If in e, v is the initial value of A and no WRITE precedes the READ, then the initial value of A is also v_0 and the READ in e' has no preceding WRITE. Thus v is a proper value to RETURN in e'. Therefore algorithm A' satisfies the regular property.

A' trivially satisfies the symmetric property because A satisfies the symmetric property, and $R_{A'}(k-1) \leq m-p$. ■

Lemma 5.4 $R(k-1) \leq R(k) - 1$.

Proof Choose any $A \in SYM(k)$ with $R_A(k) = R(k) = m$. Let C be a reachable configuration of A. Since A is a one-write algorithm, C has a neighbor D such that $f_A(D) \neq f_A(C)$. By Lemma 5.3 with $p = 1$, there exists an $A' \in SYM(k-1)$ with $R_{A'}(k-1) \leq m-1$. Thus $R(k-1) \leq m-1$. ■

Lemma 5.5 *If k is not a power of 2, then $R(k-1) \leq R(k) - 2$.*

Proof Choose any $A \in SYM(k)$ with $R_A(k) = R(k) = m$. If we can show that there exists a reachable configuration C and some $w \neq f_A(C)$ with at least two neighbors D_1 and D_2 such that $f_A(D_1) = f_A(D_2) = w$, then the result would follow from Lemma 5.3, substituting 2 for p. The rest of this proof is devoted to showing that such a configuration exists. Suppose in contradiction that for every reachable configuration C and every $w \neq f_A(C)$, C has at most one neighbor D with $f_A(D) = w$.

Claim 5.1 *For any reachable C, f_A maps all nonreachable neighbors of C to $f_A(C)$.*

Proof Suppose in contradiction that C has one nonreachable neighbor E such that $f_A(E) \neq f_A(C)$. C already has a reachable neighbor D with $f_A(D) = f_A(E)$ because A is a one-write algorithm. This means that C has at least two neighbors mapped by f_A to $f_A(E)$, a contradiction.

End of Claim

Claim 5.2 *All configurations are reachable.*

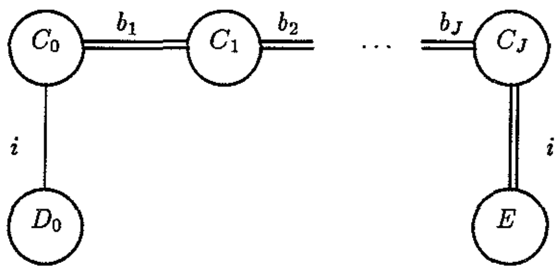


Figure 13: Relationships Among the Configurations in the Chain from C_0 to E

Proof Suppose in contradiction that there exists a nonreachable configuration. Then there exists a reachable configuration C_0 that has a nonreachable neighbor D_0. $f_A(D_0) = f_A(C_0)$ by Claim 5.1. Suppose D_0 and C_0 differ only in bit i. Since we are assuming that the minimum number of binary regular registers are used, there exists some reachable configuration E such that E and C_0 differ in bit i and bit i labels the last edge in some path of algorithm edges in H_A connecting C_0 and E. The length of the path from C_0 to E must be at least 2. Let the path be denoted by the bits that were changed in the path: $b_1, b_2, \dots, b_J, i$. Suppose the sequence of configurations in the path is $C_0, C_1, C_2, \dots, C_J, E$. Then C_J and E differ only in bit i. Figure 13 shows the relationships among these configurations. Double lines denote algorithm edges. Single lines denote edges which are not algorithm edges. For all j, $1 \leq j \leq J$, let D_j be the neighbor of C_j that differs from C_j in bit i. Notice that D_0 is nonreachable, and $D_J = E$, which is reachable. Since $D_0, D_1, \dots, D_J = E$ is the sequence of configurations in some path, there exists a j such that D_{j-1} is nonreachable and D_j is reachable. Figure 14 shows the relationships among C_{j-1}, C_j, D_{j-1}, and D_j. Dashed lines denote edges which are not known to be algorithm edges. Let $f_A(C_{j-1}) = v_1$. $f_A(C_j) \neq v_1$ because (C_{j-1}, C_j) is an algorithm edge. Since D_{j-1} is unreachable, $f_A(D_{j-1}) = v_1$ by Claim 5.1. Since D_{j-1} is an unreachable neighbor of reachable D_j, $f_A(D_j) = v_1$ by Claim 5.1. Thus C_j has two neighbors mapped by f_A to v_1, a contradiction.

End of Claim

Choose some $v \in V$. Let b be the number of configurations C with $f_A(C) = v$. Let B be the set of edges (C, D) such that either $f_A(C) = v$ and $f_A(D) \neq v$ or $f_A(C) \neq v$ and $f_A(D) = v$. For each configuration C such that $f(C) = v$, C has $k-1$ neighbors D with $f_A(D) \neq v$ by Claim 5.2 and the assumption made about all reachable configurations. This implies that $|B| = b(k-1)$. For each configuration C such that $f_A(C) \neq v$, C has one neighbor D with $f_A(D) = v$ by Claim 5.2 and the assumption made about all reachable configurations. This implies that $|B| = 2^m - b$. Then $2^m - b = b(k-1)$, which implies that $2^m = kb$, which

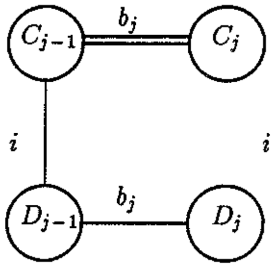


Figure 14: Relationships Among C_{j-1}, C_j, D_{j-1}, and D_j

means that k is a power of 2. This contradicts our assumption that k is not a power of 2. ■

Theorem 5.2 $R(k) > 2k - 2 - \lfloor \log k \rfloor$.

Proof We proceed by induction on k.

Basis: ($k = 4$.) $2k - 2 - \lfloor \log k \rfloor = 4$. By Lemma 5.2, $R(4) > 4$.

Inductive step: ($k > 4$.) Suppose the lemma is true for $k - 1$. Now we show that it is true for k. There are two possibilities for k. Either k is a power of 2, or k is not a power of 2.

Case 1: k is a power of 2.

$R(k) \geq R(k-1) + 1$ by Lemma 5.4

$> 2(k-1) - 2 - \lfloor \log(k-1) \rfloor + 1$, by the inductive hypothesis

$= 2k - 2 - 2 - (\lfloor \log k \rfloor - 1) + 1$, because k is a power of 2

$= 2k - 2 - \lfloor \log k \rfloor$.

Case 2: k is not a power of 2.

$R(k) \geq R(k-1) + 2$ by Lemma 5.5

$> 2(k-1) - 2 - \lfloor \log(k-1) \rfloor + 2$, by the inductive hypothesis

$= 2(k-1) - 2 - \lfloor \log k \rfloor + 2$, because k is not a power of 2

$= 2k - 2 - \lfloor \log k \rfloor$. ■

5.3 Justifying Restrictions on Readers

In this subsection we justify the restrictions that we placed on the readers by showing that general readers do not allow implementations which use fewer physical registers. Theorem 5.3 shows that any one-write algorithm can be converted to a normal form algorithm which uses no more registers. Theorem 5.4 shows that any symmetric algorithm can be converted to a symmetric algorithm using no more registers in which every reader reads each physical register at most once.

Theorem 5.3 *Any one-write algorithm A using m physical registers can be converted to a normal form algorithm A' which uses at most m physical registers.*

Proof Sketch We use algorithm transformation techniques as in the proof of Lemma 5.3. The writer's protocol in any execution of A' is based on the writer's protocol in a corresponding execution of A consisting of the same sequence of WRITEs but no READs. Each reader's protocol in any execution of A' is based on reader 1's protocol in a corresponding execution of A consisting of the same sequence of WRITEs but no other READs. ■

Theorem 5.4 *Any symmetric algorithm A using m physical registers can be converted to a symmetric algorithm A' using at most m physical registers in which every reader reads each physical register at most once.*

Proof Sketch We use algorithm transformation techniques as in the proof of Lemma 5.3. The writer's protocol in any execution of A' is based on the writer's protocol in a corresponding execution of A consisting of the same sequence of WRITEs. The reader's protocol in any execution of A' is based on the reader's protocol in a corresponding execution of A as follows. After a reader in A' reads a physical register for the first time during a READ, it makes a local copy of that register. It reads the local copy for all subsequent accesses to that physical register during the READ. ■

6 Conclusion

We have proven the existence of a one-write algorithm for implementing a k-ary regular register from binary regular registers. The algorithm we have developed uses $k(k-1)/2$ binary registers. It is optimal in the number of binary registers used with respect to all one-write algorithms satisfying the toggle property. We have also improved the lower bound on the number of binary registers required for all one-write algorithms satisfying the symmetric property from k to $2k - 1 - \lfloor \log k \rfloor$. Our lower bound proofs are modular, and they use our general technique for "fooling the reader". We have also simplified the readers and have justified the simplifications. An interesting open question is to determine tight bounds on the number of physical registers needed for symmetric algorithms and more general types of algorithms. Lemma 5.3, which is our general algorithm transformation technique, may help in obtaining tighter bounds. For example, if one can establish that $p = \Theta(\log k)$, then one can obtain a lower bound of $\Omega(k \log k)$ registers. Another interesting open question is to determine a lower bound on the number of registers a reader must read.

7 Acknowledgments

The work of the first and third authors was supported in part by NSF grant CCR-9010730 and an IBM Faculty Development Award. The work of the second author was supported in part by a UNC Board of Governors Fellowship.

References

[Blo87] Bard Bloom. Constructing Two-Writer Atomic Registers. In *Proceedings of the Sixth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 249–259, August 1987.

[BP87] James E. Burns and Gary L. Peterson. Constructing Multi-Reader Atomic Values from Non-Atomic Values. In *Proceedings of the Sixth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 222–231, August 1987.

[CKW91] Soma Chaudhuri, Martha J. Kosa, and Jennifer L. Welch. A One-Write Algorithm for Multivalued Regular Registers. Technical Report TR91-016, University of North Carolina at Chapel Hill, Department of Computer Science, March 1991.

[CW90] Soma Chaudhuri and Jennifer L. Welch. Bounds on the Costs of Register Implementations. In *Proceedings of the Fourth International Workshop on Distributed Algorithms*, September 1990. Also available as TR90-025 from the University of North Carolina at Chapel Hill.

[JSL90] Prasad Jayanti, Adarshpal Sethi, and Errol L. Lloyd. Minimal Shared Information for Concurrent Reading and Writing. Submitted for publication, August 1990.

[Lam86] Leslie Lamport. On Interprocess Communication. *Distributed Computing*, 1(1):86–101, 1986.

[LTV90] Ming Li, John Tromp, and Paul M. B. Vitanyi. How to Share Concurrent Wait-Free Variables. submitted for publication, June 1990.

[NW87] Richard Newman-Wolfe. A Protocol for Wait-Free, Atomic, Multi-Reader Shared Variables. In *Proceedings of the Sixth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 232–248, August 1987.

[Pet83] Gary Peterson. Concurrent Reading While Writing. *ACM Transactions on Programming Languages and Systems*, 5(1):46–55, 1983.

[SAG87] Ambuj K. Singh, James H. Anderson, and Mohamed G. Gouda. The Elusive Atomic Register Revisited. In *Proceedings of the Sixth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 206–221, August 1987.

[Tro89] J. T. Tromp. How to Construct an Atomic Variable. Technical Report CS-R8939, Centre for Mathematics and Computer Science, Amsterdam, October 1989.

[VA86] Paul M. B. Vitanyi and Baruch Awerbuch. Atomic Shared Register Access by Asynchronous Hardware. In *Proceedings of the Twenty-seventh Annual IEEE Symposium on Foundations of Computer Science*, pages 233–243, October 1986.

[Vid88] K. Vidyasankar. Converting Lamport's Regular Register to Atomic Register. *Information Processing Letters*, 28:287–290, 1988.