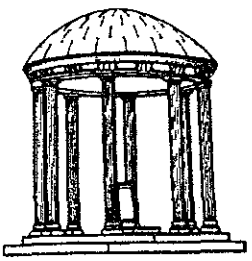
A One-Write Algorithm for Multivalued Regular Registers

TR91-016

March, 1991

Soma Chaudhur
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

A One-Write Algorithm for Multivalued Regular Registers

Soma Chaudhuri, University of North Carolina
Martha J. Kosa, University of North Carolina
Jennifer L. Welch, University of North Carolina

Abstract

This paper presents an algorithm for implementing an n-reader, k-ary, regular register using n-reader, binary, regular registers which requires only one physical write per logical write.

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. Many such implementations have been developed and proven correct [Blo87, BP87, Lam86, LTV90, NW87, Pet83, SAG87, Tro89, Vid88, VA86, CW90].

A *register* is a memory cell that supports concurrent reading and writing by a collection of processes. Reads and writes do not happen instantly; each operation starts at a certain time and ends at a later time.

Several different types of registers can be defined. The parameters that can be varied to yield different types of registers are the number of concurrent readers supported, the number of concurrent writers supported, the number of possible values the register can hold, and the strength of the consistency guarantees provided when concurrent operations occur. In this paper, we only deal with registers that support one concurrent writer and n concurrent readers ($n \geq 1$). A binary register can hold two different values. A k-ary register, where $k > 2$, can hold k different values.

Lamport [Lam86] defines three different types of consistency guarantees: safe, regular, and atomic. These consistency guarantees are listed from weakest to strongest. A read of a safe, a regular, or an atomic register returns the latest value written to the register if the read does not overlap with a write. The differences among the types of consistency guarantees appear when a read overlaps at least one write. A read of a *safe* register can return any legal value of the register. A read of a *regular* register must return either the old value of the register or a value written by one of the overlapping writes. An *atomic* register imposes a total ordering on the operations performed on the register. That is, when two reads overlap a write, the read that completes last cannot return the old value of the register if the read that completes first returns the new value of the register. In this paper, we only deal with regular registers.

Chaudhuri and Welch [CW90] studied the costs of implementing one type of register (the *logical* register) by using registers of another type (the *physical* registers). The costs they considered were the number of physical registers, the number of physical operations per logical READ, and the number of physical operations per logical WRITE[†]. They were interested in algorithms which were

[†]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.

wait-free, which means that each reader or writer can always complete a pending logical operation by performing a bounded number of actions, regardless of the actions of the other readers and writers.

Chaudhuri and Welch [CW90] proved that W, the minimum number of physical writes per logical WRITE necessary in any algorithm for implementing a k-ary regular register using binary regular registers, was such that $1 \leq W \leq \lceil \log k \rceil$. In this paper, we will show the tight bound $W = 1$ by demonstrating that a one-write algorithm for implementing a k-ary regular register using binary regular registers exists. Section 2 contains definitions which will be used in subsequent sections. Section 3 contains a specification of the algorithm. Section 4 contains a proof of correctness of the algorithm. Section 5 contains a proof that the algorithm is optimal in the number of binary regular registers if the class of algorithms to be considered is sufficiently restricted. Section 6 contains conclusions and a list of open problems.

2 Definitions

In this section, we define several terms which will be used in the specification of our algorithm and in its proof of correctness.

Assume the existence of a set V, where $|V| = k$. Let $r = C(k, 2)$. Let K_V be the complete graph with k nodes where each edge is labeled with a distinct number from the set $\{0, \ldots, r-1\}$ and each node is labeled with a distinct element from V. The **special bit set** corresponding to $v \in V$ is defined as $s(v) = \{l : l \text{ labels an edge incident to the node in } K_V \text{ corresponding to } v\}$. Since K_V is a complete graph, $|s(v)| = k - 1$. Choose some $v_0 \in V$ to be the **initial value** in V.

A **configuration** is any element of $\{0,1\}^r$. If C is a configuration, let $C[i]$ denote the i^{th} bit of C for $i \in \{0, \ldots, r-1\}$. Let the **initial configuration** be the configuration C_I such that $C_I[i] = 0$ for all $i \in \{0, \ldots, r-1\}$. Let C be a configuration. For each $v \in V$, 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 $w \neq v_0$.

Let $f : \{0,1\}^r \rightarrow V$ be the **value extraction function.** In the following, we define f. First we define f for valid configurations. Let C be a valid configuration. 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. 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. Let c be the **closest valid configuration**

Physical Registers (Bits): $X_0, \ldots, X_{r-1}$, initially $X_j = 0$, for all $j \in \{0, \ldots, r-1\}$

Reader i, $1 \leq i \leq n$: variables $x_0, \ldots, x_{r-1}$

READ(i):

for $j := 0$ to $r-1$ do $x_j :=$ read X_j endfor

RETURN($i, f(x_0 \ldots x_{r-1})$)

Writer: variables $x_0, \ldots, x_{r-1}$, initially $x_j = 0$, for all $j \in \{0, \ldots, r-1\}$, and

old, initially $old = v_0$

WRITE(v):

if $v \neq old$ then

pick i from $s(v) \cap s(old)$

write $\overline{x_i}$ to X_i

$x_i := \overline{x_i}$

$old := v$

endif

ACK

Figure 1: One-Write Algorithm

function, where $c(C)$ is defined to be the first configuration in lexicographic order in the set $\{D : D \text{ is valid and } d(C, D) \text{ is a minimum}\}$. Define $f(C)$, for C not valid, to be $f(D)$, where $D = c(C)$.

3 The Algorithm

We want to implement a k-ary regular register using binary regular registers such that only one physical write per logical WRITE is required. Let V be the value set of our logical register, where $|V| = k$. 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 f applied to the configuration obtained. 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.

4 Proof of Correctness

In this section, we prove that our algorithm implements a k-ary regular register from binary regular registers. The logical register is seen to be wait-free by inspecting the code of the read and write processes. We now show that the logical register satisfies the regular property.

A **terminal configuration** is a configuration which appears in the logical register at any point in an execution of the algorithm in which no logical WRITE is pending. Lemma 1 shows that any terminal configuration is valid and is mapped by f to the value which was written to the register by the last WRITE.

Lemma 1 *Let C be a terminal configuration resulting from a sequence of m WRITEs for the 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 $C = C_I$. C_I is valid, and $f(C) = v_0$.

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

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

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

□

A reader can read any configuration because the reader could be slow and notice traces from many WRITEs to the logical register by a fast writer. 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. Lemma 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 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 shows that an invalid configuration C agrees with its closest valid configuration C_N in the special bits corresponding to $f(C_N)$. These three lemmas are essential to the proof of Lemma 5, which shows that the reader will RETURN a correct value of the register no matter what configuration it reads.

Lemma 2 *For any time 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 time interval or occurred as the last preceding WRITE. Therefore, the lemma is true.

□

Lemma 3 *Choose any valid configurations C and D. If $f(D) = w$ and $C[i] = D[i]$ for all $i \in s(w)$, then $f(C) = w$.*

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

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

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

□

Lemma 4 *Choose any invalid configuration C. Let $D = c(C)$. Let $v = f(D)$. 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 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 in 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 configuration of the physical registers has the value v during the READ. By Lemma 2, the bits in $s(v)$ are never changed during the READ. Let D be any configuration of the bits during the READ. D is valid by Lemma 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 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, $C[i] = C_N[i]$ 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 3, $f(D) = v$, which is a contradiction.

□

The result of Lemma 5 proves the following theorem.

Theorem 6 *A one-write algorithm for implementing a k-ary regular register from binary regular registers exists.*

5 Lower Bound 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].

We can show that the upper bound of $C(k,2)$ is tight for the class of algorithms satisfying the strong toggle property (which includes our algorithm). A one-write algorithm has the **strong 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. Thus every algorithm with the strong toggle property can be represented by the complete graph on k nodes, where each node is labeled with a distinct element from V and the edge between v and w is labeled with l. 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 7 shows that $C(k,2)$ binary regular registers are necessary for any $k \geq 4$.

Theorem 7 *For all one-write algorithms A for implementing a k-ary ($k \geq 4$) regular register from binary regular registers, if A has the strong 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. Thus i labels some distinct edges (v_1, v_2) and (v_3, v_4) of G_A. 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_3, 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 f be the function from terminal configurations to V such that $f(C)$ is the value RETURNed by a logical READ that observes configuration C. Let C_1 be any terminal configuration such that $f(C_1) = v_1$. 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_3 only in bit i. By the definition of G_A, $f(C_2) = v_2$, $f(C_3) = v_3$, and $f(C_4) = v_4$. $C_1[i] \neq C_4[i]$, $C_1[j] \neq C_4[j]$, and $C_1[b] = C_4[b]$ for

all $b \in \{0, \ldots, r-1\} - \{i, j\}$. We now explain how to construct an execution of A which violates the regular property. We begin in configuration C_1. Then a logical READ begins. For the entire duration of the READ, the register is in configuration C_1, C_3, or C_4, obtained by occurrences of WRITE(v_1), WRITE(v_3), and WRITE(v_4) operations. Just before the reader reads bit i, we make sure the register is in configuration C_1, and just before the reader reads bit j, we make sure the register is in configuration C_4. Thus the reader obtains configuration C_2 and RETURNs v_2, violating the regular property.

□

A one-write algorithm has the **weak toggle property** if whenever the logical register is in configuration C such that $f(C) = v$ and the writer WRITEs w and then v, the resulting configuration is C. We have proven by brute force that 4 regular binary registers cannot implement a 4-ary regular register if the weak toggle property is assumed.

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 is optimal in the number of binary regular registers used with respect to all one-write algorithms satisfying the strong toggle property. An interesting open question is to determine a tight bound on the number of physical registers needed for more general types of algorithms.

7 Acknowledgments

This work was supported in part by NSF grant CCR-9010730 and an IBM Faculty Development Award.

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.

[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.

[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.