Bounds on the Costs of Register Implementations

TR90-025
June, 1990

Soma Chaudhuri
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.
Bounds on the Costs of Register Implementations

Soma Chaudhuri, University of Washington
Jennifer L. Welch, University of North Carolina

June 20, 1990

Abstract

A fundamental aspect of any concurrent system is how processes communicate with each other. Ultimately, all communication involves concurrent reads and writes of shared memory cells, or registers. The stronger the guarantees provided by a register, the more useful it is to the user, but the harder it may be to implement in practice. Thus it is of interest to determine which types of registers can implement which other types of registers. Algorithms for various implementations have been previously developed. These have, for the most part, concentrated on the relative computability between different types of registers. In contrast, this paper studies the relative complexity of such algorithms, by considering the costs incurred when implementing one type of register (the logical register) with registers of another type (physical registers). The cost measures considered are the number of physical registers and the number of reads and writes on the physical registers required to implement the logical register. Bounds on the number of physical operations can be easily converted to provide time bounds for the logical operations. The types of registers studied are safe vs. regular, 1-reader vs. n-readers, and binary vs. k-ary. Tight bounds are obtained on the cost measures in many cases, and interesting trade-offs between the cost measures are identified. The lower bounds are shown using information-theoretic techniques. Two new algorithms are presented that improve on the costs of previously known algorithms: the hypercube algorithm implements a k-ary safe register out of binary safe registers, requiring only one physical write per logical write; and the tree algorithm implements a k-ary regular register out of binary regular registers, requiring only \( \log k \) physical operations per logical operation. Both algorithms use novel combinatorial techniques.
1 Introduction

A fundamental aspect of any concurrent system is how processes communicate with each other. Ultimately, all communication involves concurrent accesses to shared memory cells, or registers. The stronger the guarantees provided by the shared memory, the more useful it is to the user, but the harder it may be to implement in practice. Thus it is of interest to determine which types of registers can implement which other types. Many such implementations are known [Blo87, BP87, Lam86, NW87, Pet83, SAG87, VA86].

The contribution of this paper is to study the costs of implementing one type of register (the logical register) out of registers of another type (the physical registers). Cost measures considered are the number of physical registers, and the number of operations on the physical registers used to perform the operations of the implemented register. Bounds on the number of physical operations can be used to obtain time bounds for the logical operations in terms of the time taken by the physical operations.

A register is a shared variable or memory cell that supports concurrent reading and writing by a collection of processing entities. The operations of reading and writing are not instantaneous; instead, they have duration in time, from a starting point to an ending point. Although each entity accessing a register is assumed to issue operations sequentially, operations on behalf of different entities can overlap in time.

A variety of types of registers can be defined, differing in several dimensions, including the number of concurrent readers supported, the number of concurrent writers supported, the number of values the register can take on, and the strength of the consistency guarantees provided in the presence of concurrent operations. Throughout this paper we assume there is only one writer, leaving three parameters of interest: the number of readers, the number of values, and the consistency guarantees. We distinguish between 1-reader registers and \( n \)-reader registers, for \( n > 1 \), and between binary registers and \( k \)-ary registers, for \( k > 2 \). (A \( k \)-ary register can take on \( k \) different values.)

Lamport [Lam86] defines three kinds of consistency guarantees, called safe, regular, and atomic. Roughly speaking, a read of a safe register always returns the most recent value written to the register, unless the read overlaps with a write, in which case any legal value of the register can be returned. A read of a regular register always returns the most recent value written, unless the read overlaps one or more writes, in which case it returns either the old value or one of the values written by an overlapping write. An atomic register provides the illusion, via the values returned by read
operations, that each operation happens at a single instant in time within its range, i.e., that the operations are totally ordered. In this paper, we only consider safe and regular registers.

The types of registers defined form a hierarchy of stronger and weaker definitions. For example, an \( n \)-reader, \( k \)-ary, regular register, for \( n > 1 \) and \( k > 2 \), can “implement” a 1-reader, binary, safe register \textit{a fortiori}, simply because the former has a stronger definition than the latter. Lamport [Lam86] describes implementations among safe and regular one-writer registers (as well as atomic), showing that in many cases weaker register types can implement stronger register types.

We study the costs incurred by implementations between register types. Let \( M, R, \) and \( W \) be the minima, over all implementations between two particular types of registers, of the number of physical registers, the maximum number of physical reads in a logical read, and the maximum number of physical writes in a logical write, respectively.

For implementing a \( k \)-ary safe register out of binary safe registers, we show tight bounds of \( R = \lceil \log k \rceil, W = 1, \) and \( M = \lceil \log k \rceil \). The upper bound of 1 on \( W \) is obtained from a new algorithm, which we call the \textit{hypercube} algorithm. The best previous upper bound on \( W \) was \( \lceil \log k \rceil \) [Lam86]. These three optimal bounds are not obtained simultaneously in a single algorithm, and in fact, we show some non-trivial trade-offs between the three cost measures.

For implementing a \( k \)-ary regular register out of binary regular registers, we show the tight bound that \( R = \lceil \log k \rceil \), and the bounds \( 1 \leq W \leq \lceil \log k \rceil \), and \( \max\{\lceil \log k \rceil + 1, 2(\log k - \log \log k - 2) \} \leq M \leq \min\{k - 1, n(3\log k + 68)\} \). The upper bounds on \( R \) and \( W \) are simultaneously achieved by a new algorithm, which we call the \textit{tree} algorithm. We also present some lower bounds on \( R \) and \( M \) that follow if we restrict attention to implementations that use only a small constant number of physical writes per logical write.

Our results for binary to \( k \)-ary implementations are summarized in Tables 1 and 2. Table 1 gives the bounds when all algorithms are considered. Table 2 gives the bounds when certain classes of algorithms are considered, as specified by the column labeled \( S \)—namely, 1-write algorithms, \( c \)-write algorithms, and \( \lceil \log k \rceil \)-register algorithms.

For the case of implementing an \( n \)-reader register out of 1-reader registers (either safe or regular), tight bounds are \( R = 1, W = n, \) and \( M = n \). The upper bounds are from [Lam86].

For the case of implementing a binary regular register out of a binary safe register, tight bounds are \( R = 1, W = 1, \) and \( M = 1 \). The upper bounds are from [Lam86].

All of the lower bounds mentioned above are new. Little previous work has been done concerning lower bounds or trade-offs for register implementations. The only such previous result we are aware
of is in [Lam86], where it is shown that in any implementation of an atomic register using regular registers, a read of the logical register must involve a write to a physical register.

In Section 2 we present our model and some results that are true for all implementations. The bulk of the paper concerns implementing $k$-ary registers out of binary registers: Section 3 considers safe registers and Section 4 considers regular registers. Section 5 discusses implementing $n$-reader registers out of 1-reader registers, and Section 6 discusses implementing regular registers out of safe registers. We conclude in Section 7 with some open questions.

1top row if $k$ a power of 2, bottom row if $k$ not a power of 2
2 Preliminaries

In this section, we give formal definitions for the types of registers that we will study (n-reader, k-ary, safe and regular), describe the rules we impose on implementing one type of register with another, and define the cost measures we will use. Then we present some definitions and lemmas that are true for implementations between any types of registers.

2.1 Model

We model system components using a state machine whose state transitions are labeled with actions. If there is a transition from a state labeled with an action, then that action is enabled in that state. The state machine is deterministic in that every transition from a particular state is labeled with a different action. 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. A schedule of an automaton is the sequence of actions extracted from an execution.

We model an n-reader, k-ary safe (or regular) register by an automaton $X$ as follows. Let $V$ be the value set of the register with $|V| = k$ and initial value $v_0 \in V$. Let $N$ be a set of size $n$ identifying the $n$ readers. The actions of $X$ are $\{\text{read}(i) : i \in N\} \cup \{\text{write}(v) : v \in V\} \cup \{\text{return}(i,v) : i \in N, v \in V\} \cup \{\text{ack}\}$. These are the start and finish of the read and write operations on the register: a read is terminated by a return and a write by an ack; the $i$ parameter of a read identifies the particular reader.

We restrict the register automaton $X$ to have the following property. Every schedule $\alpha$ of $X$ is well-formed, meaning that for all $i \in N$, the restriction of $\alpha$ to reads and returns for $i$ consists of alternating reads and returns, beginning with a read, and the restriction of $\alpha$ to writes and acks consists of alternating writes and acks, beginning with a write. (This models the sequential nature of the individual processing entities that access the register.) Given a sequence $\alpha$, each read($i$) instance and the following return($i,v$) instance constitute an operation, and the same for each write($v$) instance and the following ack. The two members of the same operation match each other. Every schedule $\alpha$ contains at most one unmatched write and at most one unmatched read for each $i$; these are called pending in $\alpha$.

We also require that operations can be initiated at any time, as long as well-formedness is not violated, i.e., for every schedule $\alpha$ of $X$, if no write is pending in $\alpha$, then $\alpha$ write($v$) is a schedule of
for all \(v\), and for all \(i \in N\), if no \(\text{read}(i)\) is pending in \(\alpha\), then \(\alpha\ \text{read}(i)\) is a schedule of \(X\). We call this the free initiation property.

The automaton \(X\) must return correct values for reads, where the notion of correct depends on whether the register is safe or regular. Consider any (completed) read operation in any schedule. If no write operation overlaps this read operation, then the read must return the value of the most recent preceding write; if there is no preceding write, then the read must return the initial value \(v_0\). Suppose a write does overlap the read. If the register is safe, then the read can return any value in \(V\). If the register is regular, then the read must return either the value of the most recent preceding write (or \(v_0\) if there is no such write) or some value written by an overlapping write.

The preceding intuitive discussion is now formalized. We define two kinds of possible values on finite sequences \(\alpha\), denoted \(PV_{\text{safe}}(\alpha)\) and \(PV_{\text{regular}}(\alpha)\). Suppose there is no write action in \(\alpha\). Then \(PV_{\text{safe}}(\alpha) = PV_{\text{regular}}(\alpha) = \{v_0\}\). Suppose \(\alpha = \alpha_1\ \text{write}(v)\ \alpha_2\), where there is no write action in \(\alpha_2\). If there is an ack action in \(\alpha_2\) (i.e., no write is pending), then \(PV_{\text{safe}}(\alpha) = PV_{\text{regular}}(\alpha) = \{v\}\). If there is no ack action in \(\alpha_2\) (i.e., a write is pending), then \(PV_{\text{safe}}(\alpha) = V\) and \(PV_{\text{regular}}(\alpha) = \{v\} \cup PV_{\text{regular}}(\alpha_1)\). When the superscript "safe" or "regular" on \(PV\) is clear from context, it will be dropped. We require that for any schedule \(\alpha\) of \(X\), for every read operation in \(\alpha\), the value returned is in \(PV(\alpha')\), where \(\alpha'\) is some prefix of \(\alpha\) that ends within the range of the operation.

Finally, we require that the register be wait-free: for every finite schedule \(\alpha\) of \(X\), if operation \(O\) is pending in \(\alpha\), then there is a schedule \(\alpha \pi\), where \(\pi\) is a single action, such that \(O\) is not pending in \(\alpha \pi\). This condition states that at any point in an execution at which an operation is pending, it is possible to complete the operation without waiting for any other operation to start or complete.

We now define the "rules" for implementing one type of register, the logical register, out of registers of another type, the physical registers. The type of a register specifies the number of readers supported, the number of values it can take on, and whether it is safe or regular. The building blocks for an implementation are physical registers, read processes, and write processes. There is one write process (since we are only considering 1-writer registers); the number of read processes is the number of readers to be supported by the logical register. Each read or write process is an automaton that communicates with the outside world via (logical) READ, WRITE, RETURN, and ACK actions (the actions of a logical register) and with the physical registers via (physical) read, write, return and ack actions. The read and write processes cannot communicate directly with each other. Also, in any schedule, no physical operation is pending unless a logical operation is pending, at most one physical operation is pending at any point.
We proceed more formally. Assume particular logical and physical register types with associated value sets $V$ and $V'$ respectively. Let $m$ be the number of physical registers.

A **read process** is an automaton $\text{RP}_i$, $i \in \mathbb{N}$, that has the actions $\text{READ}(i)$ and $\text{RETURN}(i, v)$, $v \in V$ (by which it communicates with the outside world), the actions $\text{read}_j(i)$ and $\text{return}_j(i, v')$, $v' \in V'$ (by which it reads registers, where $j$ ranges over the physical registers read), and the actions $\text{write}_j(v')$, $v' \in V'$, and $\text{ack}_j$ (by which it writes registers, where $j$ ranges over the physical registers written).

A **write process** is an automaton $\text{WP}$ that has the actions $\text{WRITE}(v)$, $v \in V$, and $\text{ACK}$ (by which it communicates with the outside world), the actions $\text{read}_j(0)$ and $\text{return}_j(0, v')$, $v' \in V'$ (by which it reads registers, where $j$ ranges over the physical registers read), and the actions $\text{write}_j(v')$, $v' \in V'$, and $\text{ack}_j$ (by which it writes registers, where $j$ ranges over the physical registers written).

We restrict each read process automaton $\text{RP}_i$ to have the following property. Exactly one group of actions is enabled in each state, where each action is in its own group, except that for each $j$, the set of actions $\{\text{return}_j(i, v') : v' \in V'\}$ forms a group. A $\text{READ}(i)$ transition leads to a state in which either a read$_j$, a write$_j$, or a $\text{RETURN}(i, v)$ is enabled. A $\text{RETURN}(i, v)$ transition leads to a state in which $\text{READ}(i)$ is enabled. A read$_j$ transition leads to a state in which the return$_j$ group is enabled. A write$_j$ transition leads to a state in which $\text{ack}_j$ is enabled. A return$_j$ or $\text{ack}_j$ transition leads to a state in which read$_l$, write$_l$, or $\text{RETURN}(i, v)$ is enabled, for some $l \in \mathbb{N}$ and some $v \in V$.

The write process has similar restrictions except that $\text{READ}(i)$ is replaced with the group $\{\text{WRITE}(v) : v \in V\}$, all the $\text{RETURN}(i, v)$'s are replaced with $\text{ACK}$, and the variable $i$ in the physical action names is replaced with $0$.

The restrictions specified above makes sure that no physical action is pending unless a logical operation is pending, and at most one physical operation is pending at any point.

We now describe formally how to **compose** $n$ read processes ($\text{RP}_1$ to $\text{RP}_n$), one write process ($\text{WP}$), and some number of physical registers ($X_1$ to $X_m$), in such a way as to produce another automaton $A$. First, we require that the read$_j$, return$_j$, write$_j$, and $\text{ack}_j$ actions of the read and write processes “match up” with the actions of the physical registers, i.e., for each action $\pi$ of a physical register, there is exactly one read or write process for which $\pi$ is a (physical) action, and vice versa. Note that for all $j$, the actions with subscript $j$ are actions of register $X_j$. Therefore each logical action is the action of exactly one read or write process, while each physical action is the action of one read or write process and one register. For any register $X_j$, exactly one read or
write process has the actions write\textsubscript{i} and ack\textsubscript{i}, i.e., there is a sole writer to the register.

The state set of the composition \( A \) is the cross product of the state sets of the component automata; thus each state of \( A \) is an \((n + m + 1)\)-tuple. The actions of \( A \) are the READ, WRITE, RETURN and ACK actions (the "logical" actions) and the read\textsubscript{j}, return\textsubscript{j}, write\textsubscript{j}, and ack\textsubscript{j} actions of the physical registers (the "physical" actions). Finally, we describe the transition function of \( A \).

Suppose \( s \) is a state of \( A \). We say that the logical action \( \pi \) is enabled in \( s \) if it is enabled in the state in \( s \) of the unique read or write process for which \( \pi \) is an action. We say that the physical action \( \pi \) is enabled in \( s \) if it is enabled in the states in \( s \) of both the read or write process and the register for which \( \pi \) is an action. The transition consists of each component automaton for which \( \pi \) is an action performing the action concurrently, while the remaining components do nothing.

A register implementation algorithm (or simply algorithm for short) is a composition \( A \) of \( n \) read processes (RP\textsubscript{1} to RP\textsubscript{n}), one write process (WP), and some number of physical registers (\( X_{1} \) to \( X_{m} \)) such that the composition is a logical register. This means that the schedules of \( A \), when restricted to the logical actions, satisfy the conditions for a register of the logical type. These conditions are (1) well-formedness and free initiation, which follow from the restrictions on read and write processes, (2) that logical READs RETURN values that are correct according to the possible values of the logical register, which must be ensured by the code of the read and write processes, and (3) the wait-free property, which also must be ensured by the code of the read and write processes. We actually require a stronger condition on the implementation, also called wait-free, which implies that the logical register is wait-free. This stronger condition states that each read or write process can complete a pending logical operation solely through its own actions. Formally, for every finite schedule \( \alpha \) of \( A \), if operation \( \mathcal{O} \) is pending in \( \alpha \), then there is a schedule \( \alpha\beta \), where \( \beta \) consists solely of actions of \( \mathcal{O} \)'s read/write process, such that \( \mathcal{O} \) is not pending in \( \alpha\beta \).

We need some notation to distinguish the possible values of different physical registers as well as the logical register. Let \( \alpha \) be a schedule of \( A \). For any physical register \( X \) in the composition, let \( PV_{X}(\alpha) \) be equal to \( PV(\beta) \), where \( \beta \) is the restriction of \( \alpha \) to actions of \( X \). Let \( PV_{A}(\alpha) \) be equal to \( PV(\beta) \), where \( \beta \) is the restriction of \( \alpha \) to the logical actions of \( A \).

We now define the cost measures.

Consider two register types, physical and logical, and let \( A \) be an algorithm for a physical-to-logical register implementation. Let \( M_{A} \) be the number of physical registers used in \( A \), let \( R_{A} \) be the maximum number of physical read operations performed during any logical READ in any execution of \( A \), and let \( W_{A} \) be the maximum number of physical write operations performed during any logical
WRITE in any execution of A. Given a set S of physical-to-logical register implementations, let MS be the minimum of MA over all A ∈ S, RS be the minimum of RA over all A ∈ S, and WS be the minimum of WA over all A ∈ S. Finally, let M = MS, R = RS, and W = WS, where S is the set of all physical-to-logical register implementations (for these two types). (The physical and logical register types are implicit parameters to M, R, and W.)

In the rest of this paper, we derive upper and lower bounds on M, R, and W, and tradeoffs between them, for different physical and logical register types.

These bounds on R and W can be converted into time bounds for performing logical operations as follows. Suppose we know bounds R1, Ru, W1, and Wu such that R1 ≤ R ≤ Ru and W1 ≤ W ≤ Wu. Let r be an upper bound on the time to read a physical register and let w be an upper bound on the time to write a physical register. Let s be an upper bound on the time for a read or write process to perform an action once it becomes enabled. Our upper bounds on R and W come from algorithms, all of which have the property that no logical READ involves a physical write and no logical WRITE involves a physical read. Since we assume that all physical operations are enclosed within logical operations and that only one physical operation can be pending at a time, we deduce that an upper bound on the worst case time to perform a READ of a logical register that is implemented with physical registers is Ru(r + s) + s. Similarly, an upper bound on the worst case time to perform a WRITE of a logical register that is implemented with physical registers is Wu(w + s) + s. Our lower bounds on R and W do not assume that logical READs do not involve physical writes, or that logical WRITEs do not involve physical reads, and thus they imply analogous lower bounds on the worst case times.

2.2 General Results

Fix any two physical and logical register types.

Given a finite schedule σ of an algorithm A, let the configuration of σ be the tuple of sets of possible values of the physical registers at the end of the schedule, i.e., if Xi is the i-th physical register, then the i-th element of the configuration is \( PV_{X_i}(\sigma) \). A configuration is stable if each element of the tuple is a singleton set. Thus it can be represented as \( x_1 \ldots x_m \), where \( x_i \) is the possible value of register \( X_i \) for all i. The initial configuration is the (stable) configuration of the empty schedule, consisting of the initial value of each physical register.

Let \( \overline{WO} \) (for "write-only") be the set of all schedules of A in which only WP takes steps and no physical write is pending. Let \( S = \{ C : C \) is the configuration of some \( \sigma \in \overline{WO} \} \). It is easy to see
that all configurations in $S$ are stable.

Let $WOC$ (for “write-only, completed”) be the set of all schedules of $A$ in which only WP takes steps and no logical write is pending. Let $T = \{C : C$ is the configuration of some $\sigma \in WOC\}$. It is easy to see that $T \subseteq S$. Every configuration in $T$ is defined to be a terminal configuration.

For each $i \in N$, define $L_i : S \rightarrow V$ as follows. Let $C \in S$ and $\sigma \in WO$ such that $C$ is the configuration of $\sigma$. Then $L_i(C) = v$, where

$$\sigma \text{ READ}(i) \alpha \text{ RETURN}(i, v)$$

is a schedule of $A$ such that $\alpha$ consists solely of actions of RP$_i$ and contains no RETURN. That is, $L_i$ is the logical value returned by RP$_i$ when RP$_i$ starts in its local initial state and the physical registers have the values specified in $C$. The next lemma shows that $L_i$ is well-defined, i.e., that the current configuration (values of the physical registers) and nothing else determines the value of the logical register (as perceived by RP$_i$).

**Lemma 1** For any algorithm $A$, the function $L_i$ is well-defined for all $i$.

**Proof:** Fix algorithm $A$. We must show the following two facts for any $\sigma$ and $\tau$ in $WO$ with the same configuration.

(1) There exists exactly one schedule of $A$ of the form

$$\sigma \text{ READ}(i) \beta \text{ RETURN}(i, v),$$

where $\beta$ consists only of actions of RP$_i$ and contains no RETURN.

(2) $\tau \text{ READ}(i) \beta \text{ RETURN}(i, v)$ is also a schedule of $A$.

(1) Since the read process must be wait-free, there is a schedule of the desired form. Since the configuration of $\sigma$ is stable, by the definition of a read process there is no other way to extend $\sigma$ by having RP$_i$ alone take steps.

(2) We proceed by induction. Let $\gamma = \text{ READ}(i) \beta \text{ RETURN}(i, v)$, let $\gamma$ have length $l$, and let $\gamma_j$ be the first $j$ actions in $\gamma$, $0 \leq j \leq l$. We show by induction on $j$ that

(i) $\tau \gamma_j$ is a schedule of $A$,
(ii) RP$_i$ is in the same state after $\tau \gamma_j$ as it is after $\sigma \gamma_j$, and
(iii) $PV_X(\tau \gamma_j) = PV_X(\sigma \gamma_j)$ for all physical registers $X$ in $A$. 


Basis: \((j = 0)\) \(\gamma_0 = \tau\) is a schedule of \(A\). By definition of \(\sigma\) and \(\tau\), \(\text{RP}_i\) takes no steps in \(\sigma\) or \(\tau\), and thus \(\text{RP}_i\) is in the same state, namely its initial state, after \(\gamma_0 = \tau\) as it is after \(\sigma\gamma_0 = \sigma\). Since \(\sigma\) and \(\tau\) both have the same configuration \(C\), \(PV_X(\tau\gamma_0) = PV_X(\sigma\gamma_0)\) for all physical registers \(X\).

Inductive step: \((j > 0)\) Suppose \(\tau\gamma_{j-1}\) is a schedule of \(A\), \(\text{RP}_i\) is in the same state after \(\tau\gamma_{j-1}\) as it is after \(\sigma\gamma_{j-1}\), and \(PV_X(\tau\gamma_{j-1}) = PV_X(\sigma\gamma_{j-1})\) for all physical registers \(X\). Let \(\pi\) be the \(j\)-th action in \(\gamma\), i.e., \(\gamma_{j-1}\pi = \gamma_j\). In order to show the inductive statement for \(j\), it is sufficient to show that \(\pi\) is enabled in the state of \(A\) following \(\tau\gamma_{j-1}\). Since \(\gamma\) consists entirely of actions of \(\text{RP}_i\), the following three cases are exhaustive.

Case 1: \(\pi\) is an ack action from register \(X\) to \(\text{RP}_i\). By well-formedness of \(\sigma\gamma\) and the definitions of read and write processes, there is a write action in \(\gamma_{j-1}\) to register \(X\) from \(\text{RP}_i\) with no subsequent ack. Since an ack from the physical register is enabled as soon as a write occurs, \(\pi\) is enabled in the state of \(A\) after \(\tau\gamma_{j-1}\).

Case 2: \(\pi\) is a return\((y)\) action from register \(X\) to \(\text{RP}_i\). By well-formedness of \(\sigma\gamma\) and the definitions of read and write processes, there is a read action in \(\gamma_{j-1}\) to register \(X\) from \(\text{RP}_i\) with no subsequent return. Since \(\sigma\gamma\) is a schedule of \(A\), and has no pending physical writes, \(PV_X(\sigma\gamma_{j-1}) = \{y\}\). By the inductive hypothesis, \(PV_X(\tau\gamma_{j-1}) = \{y\}\). Since a return from the physical register is enabled as soon as a read occurs, \(\pi\) is enabled in the state of \(A\) after \(\tau\gamma_{j-1}\).

Case 3: \(\pi\) is any other action of \(\text{RP}_i\). Since \(\sigma\gamma_{j-1}\) is a schedule of \(A\), \(\pi\) is enabled in the state of \(\text{RP}_i\) following \(\sigma\gamma_{j-1}\). Since \(\text{RP}_i\) is in the same state after \(\tau\gamma_{j-1}\) as it is after \(\sigma\gamma_{j-1}\), \(\pi\) is also enabled in the state of \(\text{RP}_i\) after \(\tau\gamma_{j-1}\).

The next lemma states that under certain circumstances, each \(L_i\) is equal to the possible value of the logical register.

**Lemma 2** For any algorithm \(A\), if \(\sigma\) is in \(WOC\) with configuration \(C\), then \(PV_A(\sigma) = \{L_i(C)\}\) for all \(i\).

**Proof:** Since \(\sigma\) is in \(WOC\), we know that \(\sigma = \text{WRITE}(v_1) \alpha_1 \text{ACK} \ldots \text{WRITE}(v_l) \alpha_l \text{ACK}\) for some \(v_1, \ldots, v_l\), where \(\alpha_i\), for all \(i\), consists of physical actions by the write process. It is easy to see that the logical possible value of \(\sigma\) is \(\{v_l\}\). By Lemma 1 and the safe or regular property, \(L_i(C) = v_l\).

\[\square\]
Define $L : T \rightarrow V$ to be $L(C) = L_i(C)$ for any $i$. By the previous lemma, $L$ is well-defined. It is easy to see that for each $v \in V$, there is a $C \in T$ such that $L(C) = v$.

The next lemma gives lower bounds on $R$, $W$, and $M$.

**Lemma 3** $R \geq 1$, $W \geq 1$, and $M \geq 1$.

**Proof:** Let $A$ be any algorithm.

Let $C_0$ be the initial configuration. So $L(C_0) = v_0$. Let $\sigma$ be a schedule in $\mathcal{WOC}$ with configuration $C$ such that $L(C) = v$ for some $v \neq v_0$. By Lemma 1, $C_0 \neq C$, since $v_0 \neq v$. Thus $\sigma$, and hence a logical WRITE, contains a physical write. Since $A$ was chosen arbitrarily, $W \geq 1$.

The fact that $M \geq 1$ follows from the fact that $W \geq 1$.

We show $R \geq 1$. We assume $R_A = 0$ and get a contradiction, which implies $R \geq 1$ since $A$ was chosen arbitrarily. There exists at least one schedule of $A$ of the form

$$\text{WRITE}(v) \alpha \text{ACK READ}(1) \beta \text{RETURN}(1,v),$$

where $v \neq v_0$, $\alpha$ consists solely of actions of WP and contains no ACK, and $\beta$ consists solely of actions of RP$_1$ and contains no RETURN. An easy induction shows that, since $\beta$ contains no return action (recall $R_A = 0$),

$$\text{READ}(1) \beta \text{RETURN}(1,v)$$

is a schedule of $A$, violating the safe or regular property since $v \neq v_0$.

---

### 3 $k$-ary Safe Register From Binary Safe Registers

We consider the problem of implementing an $n$-reader, $k$-ary, safe register out of $n$-reader, binary, safe registers, for any $n \geq 1$, where $k > 1$. Subsection 3.1 is devoted to proving tight, independent bounds on $R$, $W$ and $M$. In Subsection 3.2, we give some trade-offs between these measures. In particular, we show that the independent bounds are not achievable simultaneously. Let the value set of the logical register be $V = \{0, \ldots, k - 1\}$.
3.1 Independent Bounds

Theorem 4 $R = \lceil \log k \rceil$, $W = 1$, and $M = \lceil \log k \rceil$.

Proof: The upper bounds on $R$ and $M$ follow from the binary representation algorithm in [Lam86] described below. The upper bound on $W$ follows from our hypercube algorithm presented below. The lower bound on $W$ follows from Lemma 3.

We now show the lower bound on $M$. Choose any algorithm $A$. For each $v \in V$, let $C_v$ be an element of $T$ such that $L(C_v) = v$. By Lemma 1, if $v \neq w$, then $C_v \neq C_w$. Since there are $k$ distinct $C_v$'s and each is a bit string of the same length, the length of each bit string must be at least $\lceil \log k \rceil$. Thus $M_A \geq \lceil \log k \rceil$. Since $A$ was chosen arbitrarily, $M \geq \lceil \log k \rceil$.

We now show the lower bound on $R$. For each $v \in V$, there is a schedule $\sigma_v$ of $A$ of the form

\[
\text{WRITE}(v) \alpha_v \text{ ACK } \text{READ}(1) \beta_v \text{ RETURN}(1, v),
\]

where $\alpha_v$ consists solely of actions of WP and contains no ACK, and $\beta_v$ consists solely of actions of $RP_1$ and contains no RETURN.

By the definition of read processes, for all distinct $v$ and $w$, $\beta_v \neq \beta_w$ and the maximal common prefix of $\beta_v$ and $\beta_w$ is immediately followed by a return(0) action from some physical register $X$ in $\beta_v$ and by a return(1) action from $X$ in $\beta_w$ (or vice versa). I.e., $RP_1$ does the same thing in $\beta_v$ and $\beta_w$ until it reads a different value. Let $\gamma_v$ be the sequence of physical values read in $\beta_v$, for all $v$.

Thus, if $v \neq w$, then the sequence $\gamma_v$ of physical values read in $\beta_v$ is not equal to the sequence $\gamma_w$ of physical values read in $\beta_w$. There are $k$ distinct sequences of physical values corresponding to the $\gamma_v$'s, i.e., $k$ binary strings. Thus at least one string, say that corresponding to $\gamma_v$, must have length at least $\lceil \log k \rceil$, implying that $\beta_v$ contains at least $\lceil \log k \rceil$ physical reads.

Thus $R_A \geq \lceil \log k \rceil$. Since $A$ was chosen arbitrarily, $R \geq \lceil \log k \rceil$.

The binary representation algorithm in [Lam86] implements an $n$-reader, $k$-ary, safe register out of $\lceil \log k \rceil$ $n$-reader, binary, safe registers. The write process writes the binary representation of the logical value into the physical registers. Each read process reads all the physical registers and returns the logical value whose binary representation was read, as long as the value is less than $k$. 

\[\Box\]
Otherwise, it returns any value less than $k$. This algorithm implies that $R \leq \lceil \log k \rceil$, $W \leq \lceil \log k \rceil$, and $M \leq \lceil \log k \rceil$. By Theorem 4, the number of registers and number of physical reads in the binary representation algorithm are both optimal.

The **unary representation algorithm** presented next shows that $W \leq 2$. There are $k - 1$ physical registers, $X_1, \ldots, X_{k-1}$. Logical value 0 is represented when all registers are 0. Logical value $v \neq 0$ is represented when $X_v$ is 1 and the other registers are 0. Each read process reads registers $X_1, X_2, \text{etc.}$, in order, until reading a 1, and RETURNs logical value $v$, where $X_v$ is the register that returned 1. To WRITE logical value $v$, the write process writes 0 to $X_w$, where $w$ is the old value of the logical register, and writes 1 to $X_v$.

Next we describe our new **hypercube algorithm**, which shows that $W \leq 1$. For now, assume that $k$ is a power of 2. Later we will show how to remove this restriction. We define a function $f : \{0, 1\}^{k-1} \rightarrow V$ for use in the algorithm. For positive integer $i < k$, let $\text{bin}(i)$ be the binary representation of $i$ in $\log k$ bits. For $x \in \{0, 1\}^{k-1}$, let $x_i$ be the $i$th bit of $x$, i.e., $x = x_1 x_2 \ldots x_{k-1}$. For all $x \in \{0, 1\}^{k-1}$, we define $f(x)$ to be the element of $V$ whose binary representation is:

$$x_1 \circ \text{bin}(1) \oplus x_2 \circ \text{bin}(2) \oplus \cdots \oplus x_{k-1} \circ \text{bin}(k-1),$$

where $\oplus$ represents exclusive-or and $\circ$ represents multiplication. Since each $x_i$ is either 0 or 1 and each $\text{bin}(i)$ consists of $\log k$ bits, this expression consists of $\log k$ bits and thus represents a value in the range 0 to $k - 1$, i.e., a value in $V$.

**Hypercube Algorithm:**

*Physical Registers:* $X_1, \ldots, X_{k-1}$, initially $X_j = 1$ iff $j = v_0$, for all $j$

*Read Process* $R_P_i$, $1 \leq i \leq n$: variables $x_1, \ldots, x_{k-1}$

READ($i$):

```plaintext
for $j := 1$ to $k - 1$ do $x_j := \text{read } X_j$ endfor
RETURN($f(x_1 \ldots x_{k-1})$)
```

*Write process* $W_P$: variables $x_1, \ldots, x_{k-1}$, initially $x_j = 1$ iff $j = v_0$, for all $j$

WRITE($v$):

```plaintext
if there exists $j$ such that $f(x_1 \ldots x_{j-1} \overline{x_j} x_{j+1} \ldots x_{k-1}) = v$ then
write $\overline{x_j}$ to $X_j$
$x_j := \overline{x_j}$
endif
ACK
```

1
We notice an interesting relationship between the correctness of the hypercube algorithm and coloring the nodes of a \((k - 1)\)-dimensional hypercube with \(k\) colors such that each node has a neighbor with each of the \(k - 1\) colors other than its own. The following definition and lemmas formalize this idea. (Nodes are labeled with \((k - 1)\)-bit strings, the colors are elements of \(V\), and the function is the coloring.)

A function \(g\) is said to have the **rainbow-coloring property** if \(g: \{0, 1\}^{k-1} \rightarrow V\) such that for all \(x \in \{0, 1\}^{k-1}\), and for all \(v \in V\), if \(v \neq g(x)\), then there exists \(y \in \{0, 1\}^{k-1}\) such that \(v = g(y)\) and \(x\) and \(y\) differ in exactly one bit.

**Lemma 5** If function \(f\) has the rainbow-coloring property, then the hypercube algorithm is correct.

**Proof:** Clearly WP is a write process and each \(RP_i\) is a read process. Obviously the composition, \(A\), has the appropriate actions and satisfies the well-formed and free initiation properties.

We show the wait-free property. Since each physical register is wait-free, inspecting the code shows that any pending logical operation can obviously be completed using only steps of the operation's read/write process.

We show that logical READs return correct values. Let

\[
\alpha_1 \text{READ}(i) \alpha_2 \text{RETURN}(i, v)
\]

be a schedule of \(A\), where \(\alpha_2\) contains no \(\text{RETURN}(i, *)\).

Suppose there is a pending logical WRITE in \(\alpha_1 \text{READ}(i) \beta\), where \(\beta\) is any prefix of \(\alpha_2\). By the safe property, \(v\) can be any value in \(V\). Since \(f\) has the rainbow-coloring property, no matter what \(x_i\)'s \(RP_i\) obtains from the physical registers in \(\alpha_2\), \(f(x_1 \ldots x_{k-1})\) is in \(V\). Since \(v = f(x_1 \ldots x_{k-1})\), the value returned is correct.

Suppose there is no pending logical WRITE in \(\alpha_1 \text{READ}(i) \beta\) for any prefix \(\beta\) of \(\alpha_2\). Since the values of the physical registers are unchanged after \(\alpha_1\), \(RP_i\) reads the configuration \(C\) of \(\alpha_1\) during \(\alpha_2\) and \(\text{RETURNs} v = f(C)\). In order to show that \(v\) is the correct value to \(\text{RETURN}\), according to the safe property, it is enough to show that \(f(C)\) is the possible value of the logical register after \(\alpha_1\), i.e., that \(PV_A(\alpha) = \{f(C)\}\).

We proceed by induction on the number of logical WRITEs in \(\alpha_1\). Let \(\gamma_j\) be the maximal prefix of \(\alpha_1\) that contains exactly \(j\) WRITE actions and their matching ACK actions. We show that \(\{f(C_j)\} = PV_A(\gamma_j)\), where \(C_j\) is the configuration of \(\gamma_j\).
Basis: \( j = 0 \). Since there is no physical write in \( \gamma_0 \), \( C_0 \) is the initial configuration, in which \( X_j = 1 \) if and only if \( j = v_0 \). By definition of \( f \), \( f(C_0) = v_0 \), which is the logical possible value of \( \gamma_0 \).

Inductive Step: \( j > 0 \). Suppose the inductive hypothesis is true for \( j - 1 \). Note that

\[
\gamma_j = \gamma_{j-1} \text{ WRITE}(u) \beta_1 \text{ ACK} \beta_2
\]

for some \( u \), where \( \beta_1 \) and \( \beta_2 \) are sequences containing no WRITE or ACK actions. Thus the logical possible value of \( \gamma_j \) is \( u \). Since \( f \) has the rainbow-coloring property, there is a neighbor \( D \) of \( C_{j-1} \) such that \( f(D) = u \). Let \( l \) be the bit in which \( D \) and \( C_{j-1} \) differ. It is easy to see that WP keeps track of the current configuration and correctly computes \( l \). Then WP performs the physical write that changes the configuration to \( D \). Since no further physical writes occur in \( \gamma_j \), \( D = C_j \). Thus \( f(C_j) = u \).

\[\Box\]

**Lemma 6** The function \( f \) defined for the hypercube algorithm (when \( k \) is a power of 2) has the rainbow-coloring property.

**Proof:** The following two facts together show that \( f \) has the rainbow-coloring property.

- For all \( x, y \in \{0, 1\}^{k-1} \) which differ in exactly one bit, \( f(x) \neq f(y) \).
- For all \( x, y, z \in \{0, 1\}^{k-1} \) such that \( y \neq z \) and \( y \) and \( z \) both differ from \( x \) in exactly one bit, \( f(y) \neq f(z) \).

We prove the first fact. Let \( x \) and \( y \) differ in bit \( i \). Then \( f(x) \oplus f(y) = \text{bin}(i) \). Since \( \text{bin}(i) \neq 0^{\log k} \), this implies that \( f(x) \neq f(y) \). The second fact can be proved similarly. Let \( x \) and \( y \) differ in bit \( i \), and let \( x \) and \( z \) differ in bit \( j \). Then \( y \) and \( z \) differ in exactly two bits, bits \( i \) and \( j \). Then \( f(y) \oplus f(z) = \text{bin}(i) \oplus \text{bin}(j) \). Since \( i \neq j \), \( \text{bin}(i) \neq \text{bin}(j) \) and, therefore \( f(y) \neq f(z) \).

\[\Box\]

Figure 1 illustrates how our algorithm works in the simple case where \( k = 4 \). Our hypercube is then a 3-dimensional cube, whose vertices can be colored with 4 colors, \( r \), \( b \), \( g \) and \( y \). Note that the coloring satisfies the rainbow-coloring property.
Combining Lemmas 5 and 6 shows that the hypercube algorithm is a one-write algorithm (using \( k - 1 \) registers) if \( k \) is a power of 2. To obtain a one-write algorithm for values of \( k \) that are not powers of 2, we modify the power-of-2 hypercube algorithm for \( m - 1 \) physical registers, where \( m = 2^\lceil \log k \rceil \), i.e., \( m \) is the smallest power of 2 larger than \( k \). The modification is to change the RETURN statement to be \( \text{RETURN}(\min\{k - 1, f(x_1 \ldots x_{m-1})\}) \). This implementation of a \( k \)-ary register by binary registers will not cause the binary registers to take on all possible \( 2^{m-1} \) values, i.e., no stable configuration of the algorithm will be mapped to a value that is out of the range of the logical register. However, a slow read process, which overlaps a number of writes, might (spuriously) observe a configuration corresponding to a value larger than \( k - 1 \), thus necessitating the modification. Thus we have shown the following theorem.

**Theorem 7** The hypercube algorithm is correct.

### 3.2 Trade-Offs

We now consider trade-offs between the three cost measures. Theorem 8 concerns bounds on \( M \) and \( R \) for 1-write algorithms. Theorem 13 and Theorem 14 concern bounds on \( M \) and \( R \) for \( c \)-write algorithms, i.e., algorithms that use a small bounded number of physical writes per logical WRITE. Theorem 15 concerns bounds on \( W \) for algorithms that use \( \lfloor \log k \rfloor \) physical registers. Theorem 17 presents bounds on the costs of algorithms that are a hybrid of the binary and unary representation algorithms.
Theorem 8 Let $S$ be the set of algorithms $A$ such that $W_A \leq 1$. Then $R_S = k - 1$, $M_S = k - 1$ if $k$ is a power of 2, and $k \leq M_S \leq 2^{\lfloor \log k \rfloor} - 1$ if $k$ is not a power of 2.

Proof: All the upper bounds follow from the hypercube algorithm.

We now show the lower bounds. Choose $A \in S$. Let $C_{v_0}$ be the initial configuration. Then $L(C_{v_0}) = v_0$. For all $v \neq v_0$, let $C_v$ be the configuration of a schedule in WOC of the form

\[
\text{WRITE}(v) \; \alpha_v \; \text{ACK},
\]

where $\alpha_v$ contains no ACK. Lemma 2 implies that $L(C_v) = v$. Lemma 1 implies that for all $v \neq v_0$, $C_v \neq C_{v_0}$. Since $\alpha_v$ only contains one physical write, $C_{v_0}$ and $C_v$ differ in a single bit, say that for physical register $X_v$. Lemma 1 implies that for all distinct $v$ and $w$ (not equal to $v_0$), $C_v \neq C_w$. Thus $C_{v_0}$ differs from each $C_v$ in a different bit, i.e., $X_v \neq X_w$.

Since there are $k - 1$ choices for $v \neq v_0$, there are at least $k - 1$ physical registers. Since $A$ was chosen arbitrarily, $M_S \geq k - 1$. The improved lower bound of $k$ for $M_S$ when $k$ is not a power of 2 follows from Lemmas 9 and 10 below.

To show $R_S \geq k - 1$, we assume that $R_A < k - 1$ and get a contradiction; since $A$ was chosen arbitrarily, the result follows. Consider the schedule

\[
\text{READ}(1) \; \beta \; \text{RETURN}(1, v_0),
\]

where $\beta$ consists solely of actions of RP$_1$ and contains no RETURN. $\beta$ contains a sequence of less than $k - 1$ physical reads. Let $X_v$ (as defined above) be one of the physical registers not read in $\beta$; note that $v \neq v_0$. Since $C_{v_0}$ differs from $C_v$ in the value of register $X_v$ and nowhere else, an easy induction shows that

\[
\text{WRITE}(v) \; \alpha_v \; \text{ACK} \; \text{READ}(1) \; \beta \; \text{RETURN}(1, v_0)
\]

is a schedule of $A$, violating the safe condition since $v \neq v_0$.

\[\square\]

We now consider the number of registers when $k$ is not a power of 2. Lemma 9, which is the converse of Lemma 5, shows that the existence of a function with the rainbow-coloring property is necessary for the existence of a one-write algorithm using $k - 1$ registers. Lemma 10, which is the converse of Lemma 6, shows that when $k$ is not a power of 2, no function with the rainbow-coloring property can exist. Together, these two lemmas imply that if $k$ is not a power of 2, then any one-write algorithm must use more than $k - 1$ registers.
Lemma 9 If there is an algorithm $A$ with $W_A = 1$ and $M_A = k - 1$, then there exists a function with the rainbow-coloring property.

Proof: We show that $L$ has the rainbow-coloring property. We know $T$ is not empty. Choose any configuration $C \in T$. Let $L(C) = v$ and let $\sigma$ be a schedule in $WOC$ with configuration $C$. For all $w \neq v$, there is a schedule in $WOC$ of the form

$$\sigma \text{ WRITE}(w) \alpha_w \text{ ACK},$$

with configuration $C_w \in T$, where $\alpha_w$ contains no ACK.

By Lemma 2, for all $w \neq v$, $C_w \neq C$, and for all distinct $w$ and $u$ (not equal to $v$), $C_w \neq C_u$. Since there is at most one physical write in $\alpha_w$, each $C_w$ differs from $C$ in exactly one bit and no two distinct $C_w$ and $C_u$ differ from $C$ in the same bit. Since there are $k - 1$ $C_w$’s, every neighbor of $C$ is in $T$. We now show that $L : \{0, 1\}^{k-1} \rightarrow V$, by showing that $T = \{0, 1\}^{k-1}$. Clearly, $T \subseteq \{0, 1\}^{k-1}$.

Suppose in contradiction that $T \neq \{0, 1\}^{k-1}$. Then there exist $B, D \in \{0, 1\}^{k-1}$ such that $B \in T$ and $D$ is not in $T$, where $B$ and $D$ differ in a single bit. This contradicts our previous statement that all neighbors of $B$ are in $T$. Therefore, $T = \{0, 1\}^{k-1}$.

For any $C$ in $T$, and for all $w \neq v = L(C)$, $C_w$ differs from $C$ in exactly one bit and $L(C_w) = w \neq v = L(C)$. Thus $L$ has the rainbow-coloring property.

Lemma 10 If $k$ is not a power of 2, then there is no function with the rainbow-coloring property.

Proof: Assume in contradiction that there is a function $f$ with the rainbow-coloring property. Since $k$ is not a power of 2, $k$ does not divide $2^{k-1}$. Since the hypercube has $2^{k-1}$ nodes and there are $k$ colors, not all colors are assigned by $f$ in equal numbers. In particular, there is a color, call it blue, such that the number of nodes colored blue by $f$ is $b < 2^{k-1}/k$. Let $B$ be the set of edges in the hypercube that have one endpoint colored blue and one endpoint not colored blue. Since each non-blue node is adjacent to exactly one blue node and there are $2^{k-1} - b$ non-blue nodes, $|B|$ must be $2^{k-1} - b$. However, since each blue node is adjacent to $k - 1$ non-blue nodes and there are $b$ blue nodes, $|B|$ must be $b(k - 1)$. The implication is that $2^{k-1} - b = b(k - 1)$, implying $2^{k-1} = kb$, which is a contradiction.
We give bounds for $M_A$ and $R_A$, with respect to $k$ and $W_A$, given that $W_A = c$. This is of interest at values of $M_A$ between $\log k$ and $k$ and small values of $W_A$. We first prove a combinatorial lemma, which will be helpful in deriving these lower bounds in Theorems 13 and 14.

**Lemma 11** Given any binary string $x$ of length $m$, if there are at least $k$ distinct strings of length $m$ which differ from $x$ in at most $c$ bits, where $c \leq (\log k)/3$, then $m \geq (c/k/2)^{1/c}$.

**Proof:** Let $x$ be a string of length $m$. The number of distinct strings of length $m$ which differ from $x$ in at most $c$ bits is

$$\binom{m}{0} + \binom{m}{1} + \binom{m}{2} + \ldots + \binom{m}{c}$$

Since we know that there are at least $k$ such distinct strings, we have the following inequality.

$$\sum_{i=0}^{c} \binom{m}{i} \geq k$$

We obtain the following upper bound on $\binom{m}{i}$, for all $i$.

$$\binom{m}{i} = \frac{m(m-1)(m-2) \ldots (m-i+1)}{i!} \leq \frac{m^i}{i!}$$

To get an upper bound on the entire summation, we need the following claim, which is taken from [Tya88]. First, we introduce some notation. Let $s_{m,j}$ denote $\sum_{i=0}^{j} \binom{m}{i}$. Let $b_{m,i}$ denote $\binom{m}{i}$.

**Claim 12** If $j \leq m/3$, then $s_{m,j} \leq 2b_{m,j}$.
Proof: We compute a lower bound for \( \frac{b_{m,j}}{b_{m,j-1}} \) larger than 2 for \( j \leq m/3 \). Therefore, for \( j \leq m/3 \), \( \frac{b_{m,j}}{b_{m,j-1}} > 2 \). The remaining proof is by induction.

**Inductive Hypothesis:** \( s_{m,j} \leq 2b_{m,j} \) for \( j \leq m/3 \).

**Basis:** For \( j = 1 \) (assume \( m \geq 3 \)), \( s_{m,0} = b_{m,0} = 1 \) and \( b_{m,1} = m \). Therefore, \( s_{m,1} = m+1 \) and \( s_{m,1} \leq 2b_{m,1} \).

**Inductive step:** Let the inductive hypothesis hold for all \( l \) such that \( l < j \leq m/3 \). We show that it holds for \( j \). By the inductive hypothesis, \( s_{m,j-1} \leq 2b_{m,j-1} \). Note that \( s_{m,j} = s_{m,j-1} + b_{m,j} \). This implies that \( s_{m,j} \leq 2b_{m,j-1} + b_{m,j} \). Also, we showed earlier that \( 2b_{m,j-1} \leq b_{m,j} \). Therefore, \( s_{m,j} \leq b_{m,j} + b_{m,j} = 2b_{m,j} \).

The above claim holds for \( j = c \) since we know that \( m \geq \log k \) (it takes \( \log k \) bits to represent \( k \) distinct values), and this implies that \( c \leq m/3 \). Now, using the above claim and our previous upper bound for \( \binom{m}{i} \), we have

\[
\sum_{i=0}^{c} \binom{m}{i} \leq 2 \left( \binom{m}{c} \right) \leq \frac{2m^c}{c!}
\]

So, \( k \leq 2m^c/c! \) and by manipulating this inequality, we get the result \( m \geq \left( c!k/2 \right)^{1/c} \).

**Theorem 13** For all algorithms \( A \), if \( W_A = c \), where \( c \leq (\log k)/3 \), then \( M_A \geq \left( c!k/2 \right)^{1/c} \).

Proof: Given an algorithm \( A \) such that \( W_A = c \), where \( c \leq (\log k)/3 \), let \( C_{v_0} \) be the initial configuration. Then \( L(C_{v_0}) = v_0 \). For all \( v \neq v_0 \), the schedule \( \sigma_v \) of the form WRITE(\( v \)) \( \alpha_v \) ACK yields the terminal configuration \( C_v \). Since each WRITE can initiate at most \( c \) physical writes, each \( C_v \) differs in at most \( c \) bits from \( C_{v_0} \).

Since there are \( k \) values \( v \), there must be at least \( k \) terminal configurations \( C_v \) differing in at most \( c \) bits from \( C_{v_0} \). The number of registers used in the algorithm is \( M_A \). Each terminal configuration is therefore a binary string of length \( M_A \). Therefore, there are at least \( k \) strings of length \( M_A \) which differ in at most \( c \) bits from \( C_{v_0} \). Since, \( c \leq (\log k)/3 \), Lemma 11 applies, and we have the result \( M_A \geq \left( c!k/2 \right)^{1/c} \).
Theorem 14 For any algorithm $A$, if $W_A = c$, where $c \leq (\log k)/3$, then $R_A \geq (clk/2)^{1/c}$.

Proof: For any algorithm $A$, where $W_A \leq c$, consider the following schedules, for all $v$,

$$
\text{WRITE}(v) \alpha_v \text{ ACK READ}(1) \beta_v \text{ RETURN}(1, v),
$$

where $\alpha_v$ and $\beta_v$ contain only physical actions. We claim that for some $v$, $\beta_v$ initiates at least $(clk/2)^{1/c}$ physical reads. We prove this by contradiction.

Suppose, for every $v$, $\beta_v$ initiates at most $p$ physical reads where $p < (clk/2)^{1/c}$. Let $\rho_v$ be the sequence of values read, in order, on accessing any given register for the first time in $\beta_v$. Note that we don’t include values obtained from registers which have been read before or been written before in $\beta_v$. Clearly, $|\rho_v| \leq p$.

Without loss of generality, we assume that the initial configuration is the zero-vector. Therefore, the initial values of all the physical registers is 0. Since $\alpha_v$ contains at most $c$ physical writes, there can be at most $c$ 1’s in $\rho_v$. Clearly, each $\rho_v$ is distinct. Therefore, $\{\rho_v | v \in V\}$ is a set of $k$ distinct strings of length at most $p$ which differ from the zero-vector in at most $c$ bits. Since $p < (clk/2)^{1/c}$, this contradicts Lemma 11.

Therefore, for some $v$, $\beta_v$ initiates at least $(clk/2)^{1/c}$ physical reads. This gives our lower bound for $R_A$.

The next theorem states that if an algorithm uses only $\lceil \log k \rceil$ physical registers, then some logical WRITE must use at least $\lceil \log k \rceil$ physical writes.

Theorem 15 For any algorithm $A$, if $M_A \leq \lceil \log k \rceil$, then $W_A \geq \lceil \log k \rceil$.

Proof: Let $A$ be an algorithm with $M_A = \lceil \log k \rceil$. (We have already shown $M_A$ cannot be smaller.) Since the physical registers are binary, $|T| \leq 2^{\lceil \log k \rceil}$. Recall that for all $v \in V$, there is an $x \in T$ with $L(x) = v$.

Let $U$ be the subset of $T$ such that $x$ is in $U$ if and only if there is no $y \neq x$ in $T$ such that $L(y) = L(x)$. Thus for each configuration $x$ in $U$, $x$ is the only terminal configuration which has the logical value $L(x)$. 

21
Claim 16 There is an \( x \in U \) such that \( \overline{x} \in T \). (\( \overline{x} \) is the binary string that differs from \( x \) in every bit.)

Proof: Suppose there is no such \( x \). Let \( |U| = l \). Each element of \( U \) corresponds to a distinct element of \( V \), accounting for \( l \) elements of \( V \). The remaining \( k - l \) elements of \( V \) are represented among the configurations of \( T \) that are not in \( U \) and are not the inverse of an element of \( U \). There are at most \( 2^{|\log k|} - 2l \) of these configurations. There are at least two of these configurations for each remaining element of \( V \). Thus

\[
2^{|\log k|} - 2l \geq 2(k - l)
\]

\[
\implies 2^{|\log k|} \geq 2k
\]

\[
\implies |\log k| \geq \log k + 1,
\]

which is a contradiction. Thus the claim is true.

\[\square\]

Choose \( x \in U \) such that \( \overline{x} \in T \). Let \( \sigma \) be a schedule in \( WOC \) with configuration \( \overline{x} \). Suppose \( L(x) = v \). Then there is a schedule \( \tau \) in \( WOC \) of the form

\[
\sigma \, \text{WRITE}(v) \alpha \, \text{ACK},
\]

where \( \alpha \) contains no \( \text{ACK} \). The configuration of \( \tau \) must be \( x \) since \( x \in U \). Thus \( \alpha \) contains at least \( |\log k| \) writes, and \( W_A \geq |\log k| \).

\[\square\]

The binary representation algorithm yields an upper bound of \( \log k \) for \( R \), \( W \) and \( M \). The unary representation algorithm brings down the upper bound for \( W \) to 2, while pushing up the bounds for \( R \) and \( M \) to \( \Omega(k) \). This suggests a trade-off between these measures. We can construct a class of algorithms, by borrowing from both algorithms mentioned above, which have bounds on \( R_A \) and \( M_A \) varying from \( \Theta(\log k) \) to \( \Theta(k) \) and bounds on \( W_A \) varying from \( \Theta(\log k) \) to \( \Theta(1) \).

Theorem 17 For any \( m, 1 \leq m \leq k \), there is an algorithm \( A \) such that \( R_A = \Theta(\log m + k/m) \), \( M_A = \Theta(\log m + k/m) \), and \( W_A = \Theta(\log m) \).
Proof: We implement our k-ary register by combining an a-ary register and a b-ary register as follows. Let a be the smallest power of 2 which is at least as large as m, i.e., \( a = 2^{\lfloor \log m \rfloor} \). Let \( b = \lfloor k/a \rfloor \). We implement an a-ary register by the binary representation method, and a b-ary register by the unary representation method. Both these methods have been described earlier. Let the values represented by the a-ary register be in \( A = \{1, \ldots, a\} \) and the values represented by the b-ary register be in \( B = \{1, \ldots, b\} \). We obtain an ab-ary register by combining these two registers, where the ab values represented are in \( A \times B \). Note that \( ab \geq k \), so we have our k-ary register.

We consider the bounds of our combination register. The a-ary register uses \( \lfloor \log m \rfloor \) registers and \( \lfloor \log m \rfloor \) physical operations per logical operation. The b-ary register uses \( \lfloor k/a \rfloor \) registers, \( \lfloor k/a \rfloor \) physical reads per logical read, and 2 physical writes per logical write. This gives the combined bounds claimed by our theorem.

4 k-ary Regular From Binary Regular

We now shift our attention to regular registers. We would like to implement n-reader, k-ary, regular registers using n-reader, binary, regular registers. Subsection 4.1 shows our independent bounds on \( R, W, \) and \( M \). Subsection 4.2 contains our trade-off results. As before, we let \( V = \{0, \ldots, k-1\} \).

4.1 Independent Bounds

The following theorem establishes the independent bounds achieved for this problem.

Theorem 18 The implementation of n-reader, k-ary, regular registers by n-reader, binary, regular registers gives the following independent bounds:

- \( R = \lfloor \log k \rfloor \),
- \( 1 \leq W \leq \lfloor \log k \rfloor \), and
- \( \max\{\lfloor \log k \rfloor + 1, 2(\log k) - \log \log k - 2\} \leq M \leq \min\{k - 1, n(3 \log k + 68)\} \).
Proof: The lower bound for $R$ follows directly from the same result for safe registers. The lower bound for $W$ follows from Lemma 3. The lower bound for $M$ is shown in Lemmas 21 and 22 below.

The upper bounds on $R$ and $W$ appear simultaneously in the tree algorithm, presented below. However, this algorithm uses $k - 1$ physical registers. Lamport [Lam86] describes a complex composition of implementations to achieve an algorithm using $n(3\log k + 68)$ 1-reader physical registers (recall that $n$ is the number of reads for the logical register). It is unknown whether a better result, for example without the factor of $n$, is possible by taking advantage of the additional power when the physical registers are $n$-reader.

The modified unary algorithm is a simple algorithm in [Lam86] that gives upper bounds of $W \leq k$, $R \leq k$ and $M \leq k$. Given registers $X_0, \ldots, X_{k-1}$, the index of the lowest indexed register which has the value 1 determines the $k$-ary value represented. A READ operation reads $X_0, X_1, \ldots$, in order, until a 1 is returned. It subsequently RETURNs $v$, where the 1 was read from $X_v$. A WRITE($v$) operation writes 1 in register $X_v$, and then writes 0 in $X_{v-1}, \ldots, X_0$, in order.

We now present our new tree algorithm, which gives the improved bounds of $R \leq \lceil \log k \rceil$, $W \leq \lceil \log k \rceil$, and $M \leq k - 1$. The registers are the nodes in a binary tree. The tree represents a sort of binary search conducted by the READ operation to find the value written. The READ takes a path from the root to a leaf, while the WRITE follows a path starting from a leaf to the root. The path in the tree taken by the READ, along with the values it reads, uniquely defines the value read.

The tree representation of the registers is described as follows. Given any binary tree of $k$ leaves, the internal nodes of the tree correspond to the registers, while the leaves correspond to the $k$-ary values. How does the binary tree specify the algorithm? Let the leaves of the tree be labeled in some arbitrary manner by the $k$ values in $V$.

A WRITE($v$) operation writes into the set of registers which form the path between the root and the leaf labeled $v$, as follows:

- The first internal node written is the parent of the leaf labeled $v$. If the leaf node is the left/right child, the value written is 0/1.
- The $i$th internal node written is the parent of the $(i - 1)$st node. If the $(i - 1)$st node is the left/right child, the value written is 0/1.
• The last node written is the root.

A READ operation reads a set of registers which form a path from the root to a leaf labeled v, for some v. It subsequently returns v.

• The root node is the first node read.

• Suppose the ith node read has value 0/1. Then, if its left/right child is a leaf, then RETURN the value v, where v is the label of the leaf. Otherwise, the left/right child of the ith node is the (i + 1)st node read.

We just showed that any binary tree of k leaves completely specifies our algorithm. It remains to show that, for any k, a binary tree exists whose corresponding algorithm satisfies our bounds. Note that a binary tree is a tree where every node has either 0 or 2 children. Since any binary tree with k leaves must have exactly k − 1 internal nodes, our register bound of $M_A = k - 1$ is satisfied. We need to argue that $R_A \leq \lceil \log k \rceil$ and $W_A \leq \lceil \log k \rceil$. Since both READ and WRITE access registers which form a path from the root to a leaf of the tree, if the height of the tree is $h$, we have $R_A \leq h - 1$ and $W_A \leq h - 1$. (We subtract 1 from $h$ because the leaf does not correspond to a register.)

We show that there exists a tree such that $h = \lceil \log k \rceil + 1$. Since there are $k$ leaves and $k - 1$ internal nodes, our tree has a total of $2k - 1$ nodes. We know from graph theory that it is possible to construct a binary tree of $p$ nodes with height $\lceil \log(p + 1) \rceil$. So, we can construct a binary tree with $2k - 1$ nodes and height $h = \lceil \log(2k) \rceil = \lceil \log k \rceil + 1$. Substituting for $h$, we have the bounds $R_A \leq \lceil \log k \rceil$ and $W_A \leq \lceil \log k \rceil$. This gives our result.

Figure 2 illustrates a 7-ary register with value 3. The path marked on the tree corresponds to the physical registers read by a logical READ operation.

If $k$ is a power of 2, the registers and values form a complete binary tree of height $\log k + 1$. We describe the algorithm, for this special case, formally below. Let $v_m v_{m-1} \ldots v_1$ be the binary representation of the k-ary value $v$, where $m = \log k$. The root register is labeled $e$. For each register labeled with the binary string $i$, the strings $i0$ and $i1$ are the labels of its left and right children, respectively. Let the initial value of the logical register be $v_0$ with its binary representation being $v_{0,m} v_{0,m-1} \ldots v_{0,1}$. Then the initial value of the physical register labeled $v_{0,m} \ldots v_{0,p+1}$ is $v_{0,p}$, for all $p \in \{1, \ldots, m\}$. All other physical registers have initial value 0.
Figure 2: An Example Illustrating the Tree Algorithm

Tree Algorithm for $k$ a power of 2:

To WRITE(v),

for $p := 1$ to $m$ do

write $v_p$ to register $v_m \ldots v_{p+1}$

ACK

To READ,

for $p := m$ to 1 do

$v_p :=$ read register $v_m \ldots v_{p+1}$

RETURN($v_m \ldots v_1$)

Here, the log$k$ physical values read by the READ operation form the binary representation of the $k$-ary number. Clearly, the algorithm has the bounds shown.

In order to prove the correctness of the tree algorithm, we need some definitions and a lemma. We define what it means for a physical write to be before a physical read in a given schedule. We say that $w$ is before $r$, if either the physical write $w$ completely precedes the physical read $r$, or $w$ and $r$ overlap and $r$ returns the value that $w$ writes. We say that a logical READ $R$ notices a logical WRITE $W$ if there exists a physical register $s$ such that $W$ writes $s$ before $R$ reads $s$. Now, we state the following lemma.

**Lemma 19** Given any schedule of the tree algorithm, and any READ $R$ in the schedule, $R$ RETURNs the value written by the last WRITE $W$ that $R$ notices (note that there is a total order between the WRITE operations). If no such WRITE exists, $R$ RETURNs the initial value.
Proof: Let \( R \) be a READ in some schedule. Suppose \( R \) notices no WRITEs. Then every physical read \( r \) initiated by \( R \) returns the initial value of the physical register read. Therefore, \( R \) RETURNs the initial value of the logical register.

Otherwise, \( R \) notices some WRITEs. Let \( W \) be the last WRITE that \( R \) notices. Let \( s \) be the last register read by \( R \) such that \( W \) writes \( s \) before \( R \) reads \( s \). Clearly, \( R \) reads the value \( b \) written by \( W \) into \( s \). Otherwise, there is a later WRITE \( W_1 \) such that \( W_1 \) writes \( s \) and \( R \) notices \( W_1 \), which contradicts the fact that \( W \) is the last WRITE that \( R \) notices.

Without loss of generality, let \( b = 0 \). (The argument for \( b = 1 \) is identical by replacing “left” in the following discussion with “right”.)

We claim that \( s \) is the last register read by \( R \). Suppose not. Then, \( R \) next reads the register \( t \) corresponding to the left son of \( s \). Since \( W \) wrote \( b \) in register \( s \), it must have earlier written to register \( t \). This contradicts the definition of \( s \).

Now, the left son of \( s \) must be a leaf node. Let \( v \) be the label of this leaf node. Clearly, \( v \) is RETURNed by \( R \). Since \( W \) writes \( b \) into \( s \), the logical value written by \( W \) is \( v \).

\[ \square \]

Theorem 20 The tree algorithm implements a \( k \)-ary regular register using binary regular registers.

Proof: Clearly the algorithm has the wait-free property.

Given any schedule, and any READ \( R \) in that schedule, we need to prove that \( R \) RETURNs the value of one of the WRITE operations it overlaps with or the last preceding WRITE \( W_1 \). We consider two cases.

Case 1: \( R \) notices no WRITEs.
Since \( R \) reads the root node, and any WRITE must write into the root node, it follows that no WRITE completely precedes \( R \). By Lemma 19, \( R \) RETURNs the initial value, and this satisfies regularity.

Case 2: \( R \) notices some WRITEs.
Let \( W_1 \) be the last WRITE that \( R \) notices. By Lemma 19, \( R \) RETURNs the value written by \( W_1 \). We show that \( W_1 \) either overlaps with \( R \) or is the last WRITE preceding \( R \). This would satisfy regularity.
Clearly, $W_1$ cannot completely follow $R$, since, by the definition of notice, $W_1$ writes into some physical register which is subsequently read by $R$. The only other case to consider is that $W_1$ precedes another WRITE $W_2$, which completely precedes $R$. Since $W_1$ is the last WRITE that $R$ notices, $R$ does not notice $W_2$. Since $W_2$ completely precedes $R$, $R$ must read the root node after $W_2$ writes into it, which implies that $R$ does notice $W_2$. This gives a contradiction. Therefore, $W_1$ either overlaps with $R$ or is the last WRITE preceding $R$.

The tree algorithm simultaneously gives us the best bounds we have for this problem. We present our lower bounds for $M$ below. Both of the bounds we obtain are significant for different values of $k$. Figure 3 illustrates which bound is better for particular values of $k$.

**Lemma 21** $M \geq \lceil \log k \rceil + 1$.

**Proof:** Choose any algorithm $A$. We assume, for contradiction, that $M_A = \lceil \log k \rceil$. Note that the lower bound for $M$ of $\lceil \log k \rceil$, proved for safe registers, holds here as well. For all $v \in V$, there is a schedule $\sigma_v$ of $A$ in $WOC$ of the form

---

Figure 3: The Lower Bound for $m$ at Different Values of $k$
where $\alpha_v$ contains no ACK. Let $C_v$ be the configuration of $\sigma_v$; it is easy to see that $C_v$ is stable.

Choose $v \in V$. For each $w \in V$, $w \neq v$, there is a schedule $\sigma_{vw}$ in $\mathcal{WO}_C$ of the form

$$\text{WRITE}(v) \alpha_v \text{ ACK},$$

where $\alpha_v$ contains no ACK. Let $C_v$ be the configuration of $\sigma_v$; it is easy to see that $C_v$ is stable.

Let $C_{vw}$ be the configuration of $\sigma_{vw}$; it is easy to see that $C_{vw}$ is stable.

Since only WP takes steps in $\sigma_{vw}$ and physical writes are done serially, $\beta_{vw}$ goes through a sequence of stable configurations (corresponding to schedules in $\mathcal{WO}$). By Lemma 2, $L_1(C_{vw}) = w$ and $L_1(C_v) = v$. Since $w \neq v$ and $L_1$ is a function by Lemma 1, $C_{vw} \neq C_v$. Thus a stable configuration is reached in $\beta_{vw}$ that is different than $C_v$. Let $D_{vw}$ be the first such configuration. $D_{vw}$ and $C_v$ differ in a single bit, i.e., in the value of a single register.

Since there are only $\lceil \log k \rceil$ bits in each configuration, there are only $\lceil \log k \rceil$ configurations which differ in a single bit from $C_v$. Since there are $k - 1$ values in $V$ different than $v$, there exist distinct $w$ and $u$ in $V$ such that $D_{vw} = D_{vu}$. Call this configuration $D_v$. By regularity, $L_1(D_{vw}) \in \{v, w\}$ and $L_1(D_{vu}) \in \{v, u\}$. Thus $L_1(D_v) = v$.

Since $L_1(C_v) = v$, all the $C_v$'s are distinct. Since $L_1(D_v) = v$, all the $D_v$'s are distinct. It is easy to see that $C_v \neq D_w$ for all $v$ and $w$. Thus there are at least $2k$ distinct stable configurations, requiring at least $\lceil \log k \rceil + 1$ registers. Therefore, we have a contradiction.

\[ \square \]

**Lemma 22** $M \geq \lfloor 2\log k - \log \log k \rfloor - 2$.

**Proof:** Choose any algorithm $A$. Let $d$ be the number of registers used in the algorithm.

For all $v \in V$, there is a schedule $\sigma_v$ of $A$ in $\mathcal{WO}_C$ of the form

$$\text{WRITE}(v) \alpha_v \text{ ACK},$$

where $\alpha_v$ contains no ACK. Let $C_v$ be the configuration of $\sigma_v$; it is easy to see that $C_v$ is stable. Clearly, $L_1(C_v) = v$.

We claim that for any two $k$-ary values $v$ and $w$, there exist a pair of stable configurations $D_v$ and $D_w$ which differ in exactly one bit such that $L(D_v) = v$ and $L(D_w) = w$. Suppose not. Then, consider the schedule $\sigma_{vw}$ in $\mathcal{WO}_C$ of the form
\[ \sigma_v \text{ WRITE}(w) \beta_{vw} \text{ ACK}, \]

where \( \beta_{vw} \) contains no ACK. Let the configuration of \( \sigma_{vw} \) be \( D_{vw} \). The configuration of \( \sigma_v \) is \( C_v \). Note that \( D_{vw} \) is a stable configuration and \( L_1(D_{vw}) = w \). Consider the sequence of stable configurations reached by the schedule \( \sigma_{vw} \) starting from \( C_v \) and ending at \( D_{vw} \). By the assumption, there exists a stable configuration \( D_x \) in the sequence such that \( L_1(D_x) = x \) but \( x \neq v \) and \( x \neq w \). A READ starting at \( D_x \) would therefore RETURN \( x \), which violates regularity. This gives a contradiction.

Let \( c_v \) be the number of stable configurations \( C \) in \( S \) such that \( L_1(C) = v \), for each \( k \)-ary value \( v \). Let \( c = \min\{c_x | x \in V\} \), and let \( v \in V \) be such that \( c = c_v \). For each value \( w \) such that \( w \neq v \), there are stable configurations \( D_v \) and \( D_w \) in \( S \) which differ in exactly one bit such that \( L_1(D_v) = v \) and \( L_1(D_w) = w \). Since each stable configuration \( C \), such that \( L_1(C) = v \), has \( d \) neighbors, and there are \((k-1)\) values \( w \), it follows that

\[ cd \geq k - 1. \]

Since there are \( k \) different values and at most \( 2^d \) possible stable configurations,

\[ ck \leq 2^d. \]

We solve the two inequalities below.

\[ c \geq \frac{k - 1}{d} \quad \text{and} \quad c \leq \frac{2^d}{k} \]

\[ \implies \frac{k - 1}{d} \leq c \leq \frac{2^d}{k} \]

\[ \implies k^2 - k \leq d \cdot 2^d \]

\[ \implies k^2/2 \leq d \cdot 2^d, \quad \text{for} \quad k \geq 2. \]

\[ \implies 2(\log k) \leq d + \log d + 1. \]

The last inequality implies that \( d \geq 2(\log k) - \log \log k - 2. \)

\[ \square \]

4.2 Trade-Offs

We have the following lower bounds for \( R \) and \( M \) relating to one-write algorithms. In particular, we show that any one-write algorithm for this problem would require at least \( k \) registers. In other words, our hypercube algorithm for safe registers does not work for regular registers.
Theorem 23: For all algorithms \( A \), if \( W_A = 1 \) then \( R_A \geq k - 1 \) and \( M_A \geq k - 1 \).

Proof: The lower bound for \( R_A \) follows from the same result for safe registers. By using a similar argument, we can actually make the additional claim that every READ reads at least \( k - 1 \) distinct physical registers. We use this claim in the following proof of the bound for \( M_A \).

To show \( M_A \geq k \), suppose in contradiction that a one-write algorithm \( A \) exists which uses \( k - 1 \) registers. Then Lemma 9 carries over from the safe case, implying that the function \( L \) has the rainbow-coloring property. Let \( C_0 \) be the initial configuration; clearly, \( L(C_0) = v_0 \). Consider the following schedule \( \alpha \):

\[
\text{READ}(1) \delta \text{ RETURN}(1, v_0)
\]

where \( \delta \) consists only of physical actions taken by \( \text{RP}_1 \). We claim that \( \delta \) does not contain any physical write.

Claim 24: The sequence of actions \( \delta \) does not contain a physical write.

Proof: Suppose \( \delta \) does contain a physical write, i.e., \( \delta = \delta_1 \text{ write}_i(b) \delta_2 \), where \( \delta_1 \) contains no physical write. Then, there is a schedule of the form

\[
\text{READ}(1) \delta_1 \text{ write}_i(b) \delta_2 \text{ RETURN}(1, v_0) \text{ READ}(1) \delta' \text{ RETURN}(1, v_0),
\]

where \( \delta' \) contains only physical actions. Let \( C_1 \) be the configuration that differs from \( C_0 \) only in position \( i \). Then \( L(C_1) = v \), for some \( v \neq v_0 \).

Consider the schedule

\[
\text{WRITE}(v) \gamma \text{ ACK},
\]

where \( \gamma \) contains only physical actions of \( \text{WP} \). Then \( \gamma \) consists of a single physical write, to register \( i \) (as well as possibly some physical reads). An easy induction shows that

\[
\text{READ}(1) \delta_1 \text{ WRITE}(v) \gamma \text{ ACK} \text{ write}_i(b) \delta_2 \text{ RETURN}(1, v_0) \text{ READ}(1) \delta' \text{ RETURN}(1, v_0)
\]

is a schedule, since there is no physical write in \( \delta_1 \) and the physical write within the logical WRITE is "obliterated" by \( \text{write}_i(b) \). This violates regularity because the second READ should RETURN \( v \), not \( v_0 \).
Now, we continue with the proof of the theorem. Pick two distinct registers (call them registers $i$ and $j$) which are read in schedule $\alpha$.

We define $C_1$ to be the stable configuration which differs from $C_0$ in position $j$, $C_3$ to be the stable configuration which differs from $C_0$ in position $i$, and $C_2$ to be the stable configuration which differs from $C_0$ in positions $i$ and $j$. For all $l \in \{1, 2, 3\}$, $C_l$ is a terminal configuration. Let $L(C_l) = v_l$. It is easy to verify that $v_0$, $v_1$, $v_2$ and $v_3$ are distinct values in $V$. Suppose, without loss of generality, that the initial value of both registers $i$ and $j$ is 0. Figure 4 illustrates the relation between the four configurations defined. Adjacent configurations differ in a single bit. The label on the edge between two configurations corresponds to the particular bit in which they differ.

Now, consider the following sequences of actions which can be applied at a configuration $C_{\text{start}}$ and results in the configuration $C_{\text{finish}}$. 

\[\begin{array}{cccc}
C_0 & i=0 & j=0 \\
C_3 & i=1 & j=0 \\
C_2 & i=1 & j=1 \\
C_1 & i=0 & j=1 \\
\end{array}\] 

Figure 4: Relationship Between the Four Configurations

\[\square\]
We claim that if we have a schedule $\sigma$ with the configuration $C_{\text{start}}$ and no pending WRITE, we can concatenate an appropriate sequence of actions $\beta$ (from the table above) to $\sigma$ to obtain the schedule $\sigma'$ with the configuration $C_{\text{finish}}$. The sequence $\beta$ is a single logical WRITE which consists of a single physical write (and possibly some physical reads)—thus none of the $\gamma_{ab}$'s contain any physical writes. It is easy to see that each $\beta$ exists.

We create a new schedule $\alpha'$ by taking $\alpha$ and inserting certain sequences at certain points, according to the following rules. First, we insert $\beta_01$ before READ(1), resulting in configuration $C_1$. Then, before each read $j$ of RP$_1$, if the configuration is $C_1$, we insert $\beta_{12}\beta_{23}$, resulting in configuration $C_3$. Before each read $i$ of RP$_1$, if the configuration is $C_3$, we insert $\beta_{32}\beta_{21}$, resulting in configuration $C_1$. To see that $\alpha'$ is a schedule, it is sufficient to observe that the only time the configuration changes within the schedule is when a sequence $\beta_{ab}$ is inserted. This follows from the fact, proven in Claim 24, that $\alpha$ contains no physical writes. In particular, inserting $\beta_{01}$ changes the configuration to $C_1$, inserting $\beta_{12}\beta_{23}$ changes the configuration to $C_3$, and inserting $\beta_{32}\beta_{21}$ changes the configuration to $C_1$. We can prove, by a simple induction, that the configuration reached by any prefix of schedule $\alpha'$ up to a read $i$ by RP$_1$ is always $C_1$. Similarly, the configuration reached by any prefix of schedule $\alpha'$ up to a read $j$ by RP$_1$ is always $C_3$. Therefore, $\text{read}_i$ and $\text{read}_j$ always return the value 0. It follows that $v_0$ is the value RETURNed by the READ(1) in the schedule $\alpha'$. Since, to satisfy regularity, the READ should RETURN $v_1$, $v_2$ or $v_3$, we have a contradiction.

We conclude this section with a trade-off result relating to a constant number of writes. This follows from the identical result derived in the safe case.

**Theorem 25** For all algorithms $A$, if $W_A = c$, where $c \leq (\log k)/3$, then $M_A \geq (ck/2)^{1/c}$ and $R_A \geq (ck/2)^{1/c}$.
5 \(n\)-Reader Registers From 1-Reader Registers

We consider the problem of implementing an \(n\)-reader, \(k\)-ary, safe (or regular) register, for \(n \geq 2\), out of 1-reader, \(k\)-ary, safe (or regular) registers. (The results are the same for safe as for regular).

**Theorem 26** \(R = 1, W = n, \text{ and } M = n\).

**Proof:** An algorithm in [Lam86] implies that \(R \leq 1, W \leq n, \text{ and } M \leq n\). The algorithm consists of \(n\) physical registers, \(n\) read processes and one write process. To do a logical WRITE, the write process writes the logical value into each of the \(n\) physical registers and then ACKs. To do a logical READ, the (appropriate) read process reads the value in its associated physical register and RETURNS that value.

Lemma 3 implies that \(R \geq 1\). We argue the lower bounds for \(W\) and \(M\). Choose any algorithm \(A\).

\(W \geq n\): Let \(S_i\) be the set of physical registers read by read process \(RP_i\) for all \(i\). Since the physical registers are 1-reader, the \(S_i\)'s are disjoint. We now show that some logical WRITE must write at least one physical register in each \(S_i\), implying that \(W_A \geq n\) and thus \(W \geq n\). Suppose in contradiction that no logical WRITE of \(A\) writes at least one physical register in each \(S_i\). There exists a schedule of \(A\) in \(WOC\) of the form

\[
\text{WRITE}(v) \alpha \text{ ACK},
\]

where \(v \neq v_0\) and \(\alpha\) contains no ACK. By assumption, there is some \(i\) such that \(\alpha\) contains no write to any register in \(S_i\). There exists a schedule of \(A\) of the form

\[
\text{READ}(i) \beta \text{ RETURN}(i, v_0),
\]

where \(\beta\) consists only of actions of \(RP_i\) and contains no RETURN. An easy induction shows that

\[
\text{WRITE}(v) \alpha \text{ ACK} \text{ READ}(i) \beta \text{ RETURN}(i, v_0)
\]

is a schedule of \(A\), violating the safe or regular property since \(v \neq v_0\).

\(M \geq n\): Since we just showed that some WRITE writes at least one physical register in each \(S_i\) and the \(S_i\)'s are disjoint, \(M_A \geq n\) and thus \(M \geq n\).
6 Regular Registers From Safe Registers

We can show tight bounds on $R$, $W$, and $M$, for implementing an $n$-reader, binary, regular register out of $n$-reader, binary, safe registers, for $n \geq 1$. Note that given all the preceding algorithms, this case is all that is necessary to implement any kind of regular register out of any kind of safe register — one can simply compose algorithms.

**Theorem 27** $R = 1$, $W = 1$, and $M = 1$.

**Proof:** The lower bounds follow from Lemma 3.

The upper bounds follow from an algorithm in [Lam86]. The algorithm has one physical register, $n$ read processes, and one write process. To read the logical register, the (appropriate) read process simply reads the physical register and returns the value read. To write the logical register, the write process writes the new value into the physical register if and only if the new value is different than the old value (the last value written). Since every physical write toggles the value of the (safe) physical register, the desired regular behavior for the logical register is achieved.

\[ \square \]

7 Conclusion

We have demonstrated upper and lower bounds on the number of physical registers, the number of physical reads in a logical read, and the number of physical writes in a logical write, for a variety of register implementations. In many cases, our bounds are tight. Some of our upper bounds follow from two new algorithms that we present, one for implementing a $k$-ary safe register out of binary safe registers, and another for implementing a $k$-ary regular register out of binary regular registers. We also presented several interesting trade-offs between these cost measures, for implementing $k$-ary registers out of binary registers. The bounds on the number of physical operations can be converted into bounds on the time to perform the logical operations, in terms of the time for the physical operations.

Future work includes finding such bounds for more algorithms, in particular, those involving atomic registers and multi-writer registers. We also do not yet have tight bounds on $W$ and $M$ for implementing $k$-ary regular registers out of binary regular registers. It would be interesting to see if
better bounds are possible in some cases than those obtained by composing the algorithms we have. A final question is what difference does it make, if any, if clocks are available to the read and write processes?

8 Acknowledgments

We would like to thank Brian Coan for the proof of Lemma 10. Akhilesh Tyagi suggested the combinatorial techniques required in proving Theorems 13 and 14. He also helped to obtain the proof of correctness for our tree algorithm. George Welch kindly prepared the graphical display of Figure 3. We are grateful to Hagit Attiya and Richard Ladner for many helpful comments on an earlier version of this paper. The notes [LG89] from the Distributed Computing class taught by Nancy Lynch at MIT served as a useful survey and tutorial of earlier work.

References


