1 Introduction

In this lecture, we prove the correctness of the Hypercube Algorithm to implement $k$-ary safe registers with one write, and show it works for any $k$ a power of 2. In the negative, we show that there is no 1-write algorithm for implementing $k$-ary safe registers if $k$ is not a power of 2. We also show how to implement a $k$-ary regular register from $\log k$-many safe registers, improving on our previous construction of using $k$-many safe registers. Finally, we state without proof how to implement single-reader single-writer $k$-ary atomic registers from binary atomic registers.

2 Correctness of the Hypercube Algorithm

To have a 1-write algorithm for $k$ different values, the smallest structure needed has nodes each of which represent at least $k - 1$ registers. If we have that minimum number of neighbors for each vertex, can we rainbow-color the structure? (Recall that a rainbow-coloring means that each node has a neighbor of every other color.)

Lemma 1 If a rainbow-coloring exists, then the Hypercube Algorithm is correct.

Proof: We will exhibit an algorithm that intuitively works, if a rainbow-coloring function exists.

Hypercube Algorithm

READ: read $x_j$ for all $1 \leq j \leq k - 1$

RETURN $f(X_{k-1}, \ldots, X_1)$

WRITE(v): $w := f(x_{k-1}, \ldots, x_1)$

if $v \neq w$ then write $\overline{x}_j$ to $X_j$, where $j$ is such that $f(x_{k-1}, \ldots, \overline{x}_j, \ldots, x_1) = v$.

$x_j := \overline{x}_j$

where the $x_i$ are local variables, the $X_i$ are the shared registers, $f$ is the well-defined rainbow coloring function, and $\overline{x}_i$ is the bitwise complement of the binary representation of $i$.

If $f$ is well-defined, the Hypercube can be rainbow-colored.
Now we will exhibit a rainbow-coloring function that can be efficiently computed. Each node in the hypercube is represented as $X_{k-1}, \ldots, X_1$. Each color is represented by a log $k$-bit string. We start with color(000) = 00. We color 001 by $\text{XOR}$ing with the old color and the bit that is changing, to wit: $00 \oplus 01 = 01$. Hence, color(001) = 01. Other examples:

- color(010) = $00 \oplus 10 = 10$
- color(100) = $00 \oplus 11 = 11$
- color(110) = color(100) $\oplus$ 10
  = $11 \oplus 10$
  = 01.

The $\text{XOR}$ function is commutative, associative, and has the following property: $x \oplus y = z \Rightarrow x \oplus z = y$. This equality holds true, no matter where $x$, $y$ and $z$ are placed. As a result, the coloring procedure described above is equivalent to

$$ f(x_3x_2x_1) = \bigoplus_{x_i=1} \text{bin}(i). $$

For example:

- color(101) = bin(1) $\oplus$ bin(11)
  = $01 \oplus 11$
  = 10

This is the same as before because color(000) = 00, which implies color(100) = 00 $\oplus$ 01, and color(101) = 00 $\oplus$ 11 $\oplus$ 01. In other words, $\text{XOR}$ing with 00 is the identity function, so the two formulas are the same. We did not need to assign 00 as the color of 000, but it gives us a cleaner formula:

$$ f(x_{k-1}, \ldots, x_1) = \bigoplus_{x_i=1} \text{bin}(i). $$

**Lemma 2** The function $f$ defined above has the rainbow-coloring property.

**Proof:** The $k$-dimensional hypercube has $k - 1$ neighbors for each vertex. It suffices to show:

1. For all $x, y \in \{0, 1\}^{k-1}$ that differ in exactly one bit, $f(x) \neq f(y)$.
2. For all $x, y, z \in \{0, 1\}^{k-1}$ where $y$ and $z$ each differ from $x$ in one bit but $y \neq z$, $f(y) \neq f(z)$.

Facts (1) and (2) guarantee the rainbow-coloring property. We will prove them now.
Proof of (1): Let \( x, y \) differ in bit \( i \). Then \( f(x) \oplus f(y) = \bin(i) \). Since \( \bin(i) \neq 0^{\log k} \), \( f(x) \neq f(y) \).

Proof of (2): Let \( x, y \) differ in bit \( i \). Let \( x, z \) differ in bit \( j \). Then \( f(x) \oplus f(y) = \bin(i) \), and \( f(x) \oplus f(z) = \bin(j) \), implying that \( f(y) \oplus f(z) = \bin(i) \oplus \bin(j) \neq 0^{\log k} \) because \( i \neq j \), hence \( f(y) \neq f(z) \).

This proves the lemma.

In fact, we have proved the following theorem.

**Theorem 3** If \( k \) is a power of 2, there is a 1-write algorithm to implement \( k \)-ary safe registers with \( M = k - 1 \) physical registers.

## 3 No 1-write algorithm if \( k \) is not a power of 2

**Lemma 4** There exists a 1-write algorithm for implementing a \( k \)-ary safe register from \( k - 1 \) binary safe registers, only if there is a function with the rainbow-coloring property from \( \{0, 1\}^{k-1} \to \{0, \ldots, k - 1\} \).

**Proof:** [Proof Idea:] From any node, we need to be able to reach \( k - 1 \) other configurations with one write. Identify a coloring with a configuration.

**Lemma 5** If \( k \) is not a power of 2, then there is no function \( f : \{0, 1\}^{k-1} \to \{0, 1, \ldots, k - 1\} \).

**Proof:** Suppose for contradiction that \( f \) does exist. Choose any color, say blue. Let \( b \) be the number of nodes colored blue. Let \( B \) be the set of edges that join nodes that are blue to nodes that are not blue. We count the cardinality of \( B \) in two ways. First, there are \( 2^{k-1} - b \) edges from other nodes to blue nodes. Second, there are \( b(k - 1) \) edges from blue nodes to other nodes. Therefore

\[
2^{k-1} - b = b(k - 1) \\
2^{k-1} = bk
\]

so \( k \) must be a power of 2.

## 4 More on the complexity of regular registers

We have seen a way to build \( k \)-ary regular registers from binary regular registers using an algorithm \( A \), such that \( M_A = k, R_A = k \) and \( W_A = k \). However, we can come up with an algorithm \( B \) where \( M_B = k - 1, R_B = \lceil \log k \rceil \), and \( W_B = \lceil \log k \rceil \). Instead of the unary structure of algorithm \( A \), we use the same concept, but with a branching aspect, so the number of reads and writes goes down. (There is no need in this algorithm for \( k \) to be a power of 2.) See Figure 1.
Figure 1: A tree structure to implement an 8-value regular register with 7 physical registers. In this example, the reader starts at the top, and follows the dashed path to read a 3. The writer starts below (in this case to write the value 6 into the simulated register), and changes the nodes it touches to 0,1,1, respectively. The reader interprets “0” as “move left,” and “1” as “move right.”
5 Algorithm for $k$-ary atomic registers

The presence of multiple readers does not matter for safe or regular registers. In this case, atomic registers, it does, because the value returned by an earlier reader can force a later reader to return the same value, even if both readers overlap a write. So we start with single-reader single-writer (SRSW) $k$-ary atomic registers built up from SRSW binary atomic registers.

**WRITE:** write $x_v = 1$

for $i := v - 1$ downto 0

write $x_i := 0$

ACK

**READ:** Read $x_i$ for $i := 0$ to $k - 1$ until $x_i = 1$

$v := i$

Read $x_j$ for $j := v - 1$ downto 0

if $x_j = 1$ then $v := j$

RETURN $v$

We will prove this next time by showing that linearizability holds.