1 Space of Registers

In this lecture we will learn how to construct powerful registers from simple boolean registers. As per the discussions from previous lectures, we have seen that there are a range of possible registers which are shown on a 3-dimensional space in Figure 1. The register size defines one dimension, the number of readers and writers defines another, and the registers consistency property defines the third.

![Figure 1: 3-d space of possible read - write register - based implementations](image)

2 Register Constructions

All the above registers are constructible from one another. We will study the implementation of powerful registers using simple registers. The physical registers that we will use for construction are wait-free meaning each method call finishes in a finite number of steps, independently of how its execution is interleaved with steps of other concurrent method calls. We require that the
constructed logical registers also be wait-free. This basically means that because each physical register responds to an invocation, so should the logical register.

### 2.1 MRSW Registers using SRSW Registers

![Diagram of Logical MRSW Register]

Figure 2: Logical MRSW Register

Figure 2 shows the construction of a MRSW register. Each Read Process \( RP_i \) reads from one physical register \( \chi_i \) and the Write Process \( WP \) writes to all the physical registers.

The algorithm for the logical reads and write is as follows:

**Algorithm 1** Algorithm for WRITE and READ

**WRITE** \( (v) \)

1. \( \forall i \in \{1, 2, ..., n\} \), do \( write_i(v) \) on \( \chi_i \)
2. wait for \( ack_i \) from all \( \chi_i \)
3. do \( ACK_i \)

**READ** \( i(v) \)

1. do read on \( \chi_i \)
2. wait for \( ret(v) \)
3. do \( RETURN(v) \)

**Lemma 1** If \( \chi_1, \chi_2, ..., \chi_n \) are safe registers, then the MRSW logical register is also a safe register.
Proof: If WRITE precedes READ, then the corresponding physical write will also precede the corresponding physical read, since they are encapsulated in the logical operations. Since the physical registers are safe, they will return the value of the preceding physical write. Hence READ will also return the value of the preceding WRITE. For overlapping READ and WRITE, the physical read and write may or may not overlap. Hence it can return any legal value which may or may not be correct. Therefore, the MRSW register satisfies the property of safe registers.

This proves that the logical MRSW register is also safe.

Lemma 2 If $\chi_1, \chi_2, ..., \chi_n$ are regular registers, then the MRSW logical register is also a regular register.

Proof: We consider the following cases:

- If the logical READ does not overlap any WRITE, then it is obvious that the physical read will also not overlap any write. Hence READ will return the value returned by corresponding underlying physical read, which in turn is the value written by the preceding write corresponding to the preceding logical WRITE (or initial value). Hence READ returns the value written by preceding WRITE.

- If the logical READ overlaps one or more WRITE’s overlap, we have the following 2 possibilities.
  
  1. The physical read overlaps some physical write and returns the value written by an overlapping write. If this happens then READ returns the value returned by the underlying read that returns the value of the overlapping write which will correspond to an overlapping WRITE (This is true because if write overlaps read, then WRITE overlaps READ). Thus READ returns the value of an overlapping WRITE.

  2. The physical read overlaps some physical write and returns the value written by the last preceding write. If the last preceding physical write is within an overlapping or last preceding WRITE, we are ok because in this case READ will return the value written by the last preceding or overlapping WRITE. Suppose, for contradiction, it is within an earlier WRITE, then there is another WRITE between the WRITE and the READ. In that case, there must be a physical write' between write and read, contradicting the fact that write is the last preceding physical write.

  3. The physical read does not overlap a physical write and returns the value of the last preceding write. This is true because the physical registers are regular. The argument for 2 also holds here because read returns the value of the last preceding physical write, hence READ will return the value written by the last preceding or overlapping WRITE.

As the constructed register satisfies all properties of regular registers, it is also regular.

3
2.2 k-ary Safe Registers from Binary Safe Registers

To construct a k-ary safe register from binary register, we need $\log_2 k$ binary registers because we can represent k-ary registers using $\log_2 k$ binary registers. For our construction, let $l = \log_2 k$.

![Diagram of k-ary Register from Binary Registers](image)

Figure 3: k-ary Register from Binary Registers

**Algorithm 2** Algorithm for WRITE and READ

**WRITE**(v)
1. ∀$i \in \{1, 2, ..., l\}$, write bit $i$ of $v$ into $\chi_i$
2. wait for ack
3. do $ACK$

**READ**$_i$(v)
1. for $i \in \{1, 2, ..., l\}$, read bit $i$ of $v$ from $\chi_i$
2. do $RETURN(v)$

This means that the WRITE call writes one bit of $\chi_i$, then proceeds to the next register. It cycles through all the registers till all bits have been written into all registers. The READ call reads all the bits of $\chi_i$ in one call. Thus if the reader comes from the time between the writes changes the first bit to the time it changes the last bit, it will read legal but incorrect values. This is explained in Figure 4.

We claim that the constructed k-ary register is safe. This is true because it satisfies the condition for safe registers i.e. a non-overlapping READ returns the value written by the most recent WRITE, and if a READ overlaps with one or more WRITE’s, then READ can return any legal value regardless of what is being written.

**Note:** $k$ does not have any relation with the number of readers $n$. 
Theorem 3 Algorithm 2 does not work for regular registers.

Proof: This is illustrated in Figure 4. Here the READ can return either 001 or 011, even though the old value is 000 and the new value written by the overlapping WRITE is 111. This violates the condition of regular registers.

2.3 k-ary Regular from Binary Regular

For regular registers, we need some redundancy in the registers for any algorithm to work. To bring redundancy in the registers, the algorithm is designed as follows.

Algorithm 3 Algorithm for WRITE and READ

\[ \text{WRITE}(v) \]

- The writer moves from left to right towards the target bit.
- It does not change any bit until it reaches the target.
- Only after writing a 1 into the target bit, it 0's out the remaining bits.

\[ \text{READ}_i(v) \]

- The reader moves from right to left (i.e., the opposite direction).
- Returns when it finds a 1.

Since the reader moves from the opposite direction, the reader may read a value when the writer is in between writing a new 1 bit and erasing the old bit. In this case, since the READ and WRITE are overlapping, it may read the value of the preceding WRITE in the worst case. Since this satisfies the condition of regular registers, the constructed k-ary register is a regular register. Figure 5 illustrates this.

2.4 k-ary Atomic from Binary Atomic

The algorithm is similar to the above case of regular registers.

Since the reader moves from the opposite direction, the reader may read a value when the writer is in between writing a new 1 bit and erasing the old bit. In this case, since the READ

![Figure 4: Example illustrating Algorithm 2](image-url)
Algorithm 4 Algorithm for WRITE and READ

\textit{WRITE}(v)

The writer moves from left to right towards the target bit.

It does not change any bit until it reaches the target.

Only after writing a 1 into the target bit, it 0's out the remaining bits.

(same as above).

\textit{READ}_i(v)

The reader moves from right to left (i.e the opposite direction).

if it encounters a 1, then moves back from left to right.

Returns the 1 that it found previously or any newly found 1.

and WRITE are overlapping, it may read the value of the preceding WRITE in the worst case.

Any newer READ, will always give either the value returned by an older READ or any other new value that has been written since. This is true because READ traverses back after reading a 1, so it will either return the same value as the previous READ, or any newer value. This satisfies the condition for atomic registers. Hence the constructed k-ary register is atomic.

2.5 Binary Regular from Binary Safe

By using a MRSW Binary Safe register, we construct a MRSW Binary Regular register as shown in Figure 6.

Lemma 4 If a READ, say R sees a 1 in \(\chi_v\), then v must have been written by an overlapping or last preceding write.
Algorithm 5 Algorithm for WRITE and READ

\textbf{WRITE}(v)
1 if \(v\) was not the last written value, then do write\((v)\)
2 wait for ack
3 do ACK

\textbf{READ}_i(v)
1 do read
2 wait for return\((v)\)
3 do RETURN\((v)\)

Proof: Consider the following.

- If \textit{READ} does not overlap with a \textit{WRITE}, then since the physical registers are safe, it will return the value written by the preceding \textit{WRITE} or the initial value if there is no preceding \textit{WRITE}.

- If \textit{READ} overlaps with a \textit{WRITE}, then we consider the following cases.

1. If the register already has a value 0/1 and the overlapping \textit{WRITE} writes a value 1/0, then the \textit{READ} can return either a 0 or a 1. Both these values are acceptable for the regular register since one of them will be the value written by a preceding \textit{WRITE} and the other one will be the value written by overlapping \textit{WRITE}.

2. If the register already has a value 0 and the overlapping \textit{WRITE} also writes a value 0, then \textit{READ} can return either a 0 or a 1 because the underlying physical register is a safe register. But as per the algorithm, \textit{WRITE} will overwrite the value in the
register only if old and new values are different. Therefore, \textit{WRITE} will not write anything and \textit{READ} will read the old value 0, satisfying the condition for regular registers that \textit{READ}, will return the value written by the preceding \textit{WRITE} or an overlapping \textit{WRITE}.

3. Point 2 applies if the old value is 1 and the new value written by \textit{WRITE} is also 1.