1 Multi-Reader Atomic Register from Single-Reader Atomic Registers

In this section we allow only single writer both for the physical registers and the logical register. Let $n$ be the number of READ processors. So it is a multi-reader-single-writer logical register. The basic building block is single-reader-single-writer atomic registers, that is built in the last lecture.

For safe or regular registers, as in previous lectures, if there are multiple readers, readers do not interfere with each other. What reader $A$ reads does not change what reader $B$ reads. There is no order between two reads as long as they interaction with the write is legal. For an overlapping write, each read just returns any possible value of the register (safe register case) or either the most recent write or the overlapping write (regular register case). So the following simple algorithm that worked for multi-reader single writer safe and regular registers.

- Writer writes into $n$ registers sequentially
- Reader $i$ reads from $x_i$ and returns the value read.

The case is fine as shown in Figure 1. Since $R_2$ can return either the most recent write $X_{v_2} = 0$, or the overlapping write $X_{v_2} = 1$, the result is legal.

However, this is not the case for atomic registers. The key difference is that for atomic registers, it asks for a linearizable sequence. Not only is legality required but also the real time precedence of any two non-overlapping operations should be preserved.

For multiple-reader when the write writes the same content into different registers sequentially, and each reader read from its own/assigned register and returns the value read, problems occur. In real time if two reads have precedence order, due to linearization requirement, the second read must be put behind the first read in linearized sequence. However, if there is an overlapping write with these two reads, it is possible that the first read reads the new value in its register, and the second read reads the old value because of the write has not updated the old value in its register. Then the sequential specification requires the second write be put before write and also before the first read. Contradiction.

So the first read has to write, to communicate with the second read about what it has got. The second write has to compare what it got from the writer and what it got from the first reader and then return the latest value. Symmetrically it also broadcasts its decision to the first reader. This way the sequence $WR_1R_2$ satisfies sequential specification/legality (both reads return the
A single overlapping write

![Diagram showing a single overlapping write.]

Figure 1: A later reader $R_2$ got an older value $X_{v_2} = 0$ than an earlier reader $R_1$ that got the new value $X_{v_1} = 1$ because the sequential write of registers by the overlapping writer $W$ takes time to finish, assuming that the writer writes identical information to each physical register. For regular register it is fine. However, as atomic logical register, $W$ can only take one instant in time axis. There is no where in this figure that the linearization point of $W$ can fit in.

most recent write in the linearized sequence $\pi$) and real time precedence between $R_1$ and $R_2$, that is, $R_1 \prec R_2$.

The root cause of the conflict is that linearization assumes each operation takes place at one spot of time. This is far from reality! It is just an image/interface seen by an innocent programmer. When conflicts between ideal state and reality occur, sometimes smart ways can be devised so that additional coordinations are arranged by participants to achieve the ideal effect.

Let us take a look at Figure 1 again. It is exactly the case discussed just now. $R_2WR_1$ is not an acceptable answer either, since while $R_2WR_1$ satisfies sequential specification of the register, the real time order of operations is not satisfied since $R_1 \prec R_2$.

Therefore readers needs to write!

**Theorem 1** In a wait-free simulation of a single-writer multi-reader register from single-writer single-reader registers, at least one reader must write.

**Proof:** Suppose, in contradiction, there exists a simulation for register $R$ where readers do not write. Let $P_w$ be the writer and $P_1$ and $P_2$ be the readers.

Suppose the initial value of $R$ is 0. Since physical registers are single reader, they can be partitioned into 2 sets, $S_1$ and $S_2$, where only $P_1$ reads $S_1$, and $P_2$ reads $S_2$.

Consider $\alpha$, the execution with a WRITE(1). $P_w$ writes to a series of physical registers $W_1, \ldots, W_k$. Each $W_i$ is either in $S_1$ or $S_2$. 

---

2
For each $i = 1, 2$, and each $j = 0, \ldots, k$, define an alternative execution $\alpha_i^j$ obtained from $\alpha$ by interposing a READ by $P_i$ after the linearization point of $W_j$ and before linearization point of $W_{j+1}$. Let $v_j^i$ be the value returned by the READ.

Let $j_i \in \{1, \ldots, k\}$ such that for all $j < j_i$, $v_j^i = 0$ and $v_{j_i}^i = 1$. Now, $j_1 \neq j_2$ since $W_{j_1}$ writes in $S_1$ and $W_{j_2}$ writes in $S_2$ and $S_1 \cap S_2 = \emptyset$. Without loss of generality, $j_1 < j_2$. Then $v_{j_1}^1 = 1$ and $v_{j_1}^2 = 0$. Let $\alpha'$ be an execution where a READ by $P_1$ followed by a READ by $P_2$ are inserted into $\alpha$ between $W_{j_1}$ and $W_{j_1+1}$. Now, $P_1$’s READ returns 1 and $P_2$’s READ returns 0, violating atomicity.

Consider the execution $\alpha_j^i$ where $i \in \{1, 2\}$ and $j \in \{0, \ldots, k\}$. Now $\alpha_0^j$ has the READ happening before any physical write, so $v_0^j = 0$. Also $\alpha_k^j$ has the READ happening after all physical writes are done, so $v_k^j = 1$, since the WRITE has completed.

Note when $S_1$ and $S_2$ are reduced to single registers respectively, the proof is just the case in Figure 1.

We need readers to write!

- Readers write to each other, to create ordering between them, into additional registers.
- Before reader returns a value, it announces the value it has decided to return.
- Reader reads not only the value written by the writer, but also values announced by other readers.
- It then returns the most recent value.

How do we choose the most recent value? Use timestamps. The writer adds a sequence number to each value it writes; requires unbounded registers!

Registers: $Val[i]$: for each $P_i$, $1 \leq i \leq n$, written by $P_w$. $Report[i, j]$: value returned by $P_i$ (written by $P_i$ and read by $P_j$).

Algorithm is clearly wait-free. Construct $\pi$ to prove linearizability.
Algorithm 1 Readers write and timestamps

1: Initially, Report\([i, j] = Val[i] = (v_0, 0), 1 \leq i, j \leq n\)
2: When read,\([R]\) occurs, \(P_r\) reads from \(R\)
3: \((v[0], s[0]) := Val[r],\) most recent value reported to \(P_r\) by writer
4: for \(i := 1\) to \(n\) do
5: \((v[i], s[i]) := Report[i, r],\) most recent value reported by \(P_i\) to \(P_r\)
6: Let \(j\) be such that \(s[j] = \max\{s[0], \ldots, s[n]\}\)
7: for \(i := 1\) to \(n\) do
8: Report\([r, i] := (v[j], s[j]),\) \(P_r\) reports to every \(P_i\)
9: Return \(r(S, v[j])\)
10: When write \((R, v)\) occurs
11: \(seq := seq + 1\)
12: for \(i := 1\) to \(n\) do \(Val[i] := (v, seq)\)
13: Acknowledge\((R)\)

- Place WRITEs in order in which they occur.
- Place READs immediately before the WRITE that follow the WRITE that has the timestamp associated with value READ returns.

By construction, \(\pi\) satisfies sequential specification.

Lemma 2 Let \(op_1\) and \(op_2\) be two logical operations in \(\alpha\) such that \(op_1\) precedes \(op_2\) in \(\alpha\). Then \(op_1\) precedes \(op_2\) in \(\pi\).

Proof: The real time order of write operations is preserved by the construction. We consider the other three cases.

Consider the read operations \(R\) by \(P_i\) that returns a value associated with time-stamp \(T\).

- Consider a write \(W\) that follows \(R\) in \(\alpha\). If \(R\) is placed after \(W\) in \(\pi\), then the write \(W'\) that generates time-stamp \(T\) is \(W\) or a later write implying that \(W'\) follows \(R\) in \(\alpha\), a contradiction.

- Consider a write \(W\) that precedes \(R\) in \(\alpha\). So \(R\) reads from \(Val[i]\), the value written by \(W\) or later write, so by the semantics of max in time-stamps, \(R\) returns a value with time-stamp at least as large as \(W\)’s. So \(R\) must be placed after \(R\) in \(\pi\).

- Consider a read \(R'\) by \(P_j\) that follows \(R\) in \(\alpha\). So \(P_j\) reads a time-stamp in \(Report[i, j]\) that is written during \(R\) or later, so it will have time-stamp \(\geq T\). So, \(P_j\)’s max time-stamp for \(R'\) will be \(\geq T\), so it will be placed after a later write than the write \(R\) is placed after (implying \(R <_\pi R'\)), or after the same write as the one \(R\) is placed after (in which case \(R'\) will be placed after \(R\) since \(R'\) ends after \(R\) in \(\alpha\).

The algorithm can be modified to use bounded registers. The single writer to multiwriter implementation also uses time-stamps between writers to figure out the latest write.