Correctness proof for multi-reader single-writer simulation

November 25, 2017

The proofs provided herein are taken from the book "Distributed Computing", by Hagit Attiya and Jennifer Welch, with some minor changes.

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

Proof: Suppose in contradiction there is such a simulation for a register $R$ in which the readers never write. Let $p_w$ be the writer, and let $p_1, p_2$ be two readers. Suppose the initial value of $R$ is 0.

Since the base-objects used by the implementation are SRSW registers, they may be partitioned into two sets, $S_1$ and $S_2$, such that only reader $p_1$ reads the registers in $S_1$, and only reader $p_2$ reads those in $S_2$.

Let $\alpha$ be the execution of a high-level write of 1 to $R$, starting in an initial configuration. $\alpha$ consists of a sequence of reads and writes to base objects. Let $w_1, \cdots, w_k$ be the series of low-level writes that appear in $\alpha$, each of which writes to a register in either $S_1$ or $S_2$.

For $i = 1, 2$ and $j = 1, \cdots, k$, define an alternative execution $\alpha^i_j$, obtained from $\alpha$ by inserting an execution of a high-level read operation by $p_i$ right after the linearization point of $w_j$. Let $v^i_j$ be the value returned by this high-level read.
Since the simulation guarantees linearizability, for each $i = 1, 2$, there exists $j_i$ between 1 and $k$ such that $v_{j_i}^i = 0$ for all $j < j_i$ and $v_{j_i}^i = 1$. That is, there is a single low-level write operation $w_{j_1}$ that causes $p_i$ to observe the simulated register as having changed its value from 0 to 1. Clearly, $j_1 \neq j_2$ holds. This is because $w_{j_1}$ writes some register in $S_1$ and $w_{j_2}$ writes some register in $S_2$, and these sets are disjoint.

W.l.o.g, assume $j_1 < j_2$ holds. Let $\alpha'$ be an execution obtained from $\alpha$ by inserting a (high-level) read by $p_1$ followed by a (high-level) read by $p_2$ right after $w_{j_1}$. Then $p_1$’s read returns 1 and $p_2$’s read returns 0 in $\alpha'$. This contradicts linearizability, since $p_1$’s read sees the newer value, yet it precedes $p_2$’s read, which sees the older value.

The following lemma proves that the simulation algorithm for multi-reader-single-writer register, shown in the presentation, is linearizable. Let $\alpha$ be an execution of the simulation algorithm. $\alpha$ is linearized to a history $\pi$ as follows. First, we put in $\pi$ all the write operations, according to the order in which they occur in $\alpha$. Next, we add the read operations to $\pi$. We consider the reads one by one, in the order of their responses in $\alpha$. A read that returns a value with time-stamp $T$ is place immediately before the write that follows (in $\pi$) the write operation that generated time-stamp $T$.

From construction, $\pi$ is a legal history.

**Lemma 2** Let $op_1$ and $op_2$ be two high-level operations in $\alpha$ such that $op_1$ ends before $op_2$ begins, then $op_1$ precedes $op_2$ in $\pi$.

**Proof:** The claim is clear for write operations.

Consider some read operation $r$ by $p_i$ that returns a value associated with time-stamp $T$. Consider a write $w$ that follows $r$ in $\alpha$. Then $w$ is associated with some time-stamp $T'$ such that $T' > T$. It follows that $r$ is placed before $w$ in $\pi$.

Consider a write $w$ that precedes $r$ in $\alpha$. Since $r$ occurs after $w$, $r$ reads from $Val[i]$ the value written by $w$ or by a later write. Thus $r$ returns a value whose associated time-stamp is generated by $w$ or a later write. It follows that $r$ is not placed before $w$ in $\pi$.  

2
Consider a read \( r' \) by \( p_j \) that follows \( r \) in \( \alpha \). By linearizability, \( p_j \) obtains a time-stamp from \( Report[i] \) during \( r' \) that is written during \( r \) or a later operation. Since time-stamps are ever increasing, and since \( r' \) associated its value with the maximum time-stamp it observed, the time-stamp associated with \( r' \) is not smaller than that associated with \( r \). Thus, as \( r' \) is considered after \( r \) while constructing \( \pi \), \( r' \) follows \( r \) in \( \pi \).