Correctness proof for atomic snapshot simulation

December 8, 2013

The proofs provided herein are based on these given in the book ”Distributed Computing”, by Hagit Attiya and Jennifer Welch, with some minor changes.

1 Atomic Simulation proofs

Lemma 1 A direct scan returns the actual value of the atomic snapshot object in the configuration immediately following the last read in the first collect of the successful double collect.

Proof: Suppose $p_i$ performs the direct scan and let $p_j$ be any other process. Consider the last iteration of the loop of lines 2-8. Let $r_1$ be the linearization point of the last read in the first collect (line 3) and let $r_2$ be the linearization point of the read of Segment[$j$] in the second collect (line 4). Since the direct scan returns the value read at $r_2$ (in line 6), we must show that no write to Segment[$j$] is linearized between $r_1$ and $r_2$. However, if such a write would have existed, Segment[$j$].ts would have been incremented and $a[j] \neq b[j]$ would hold.

Lemma 2 An indirect scan returns the view of a direct scan whose execution is enclosed within the execution of the indirect scan.

Proof: Let $p_i$ be the process that executes the indirect scan. Assume that the indirect scan borrows a view to return from process $p_j$. Thus $p_i$’s indirect scan evaluates the condition of line
7 to be true for \( j \). It follows that, since \( p_i \) started executing the scan operation, \( p_j \) executed line 2 of \texttt{update} at least twice. However, between any two consecutive executions of line 2 by \( p_j, p_j \) starts and completes a new instance of the \texttt{scan} procedure. It follows that the view returns by \( p_i \) (in line 8) is “borrowed” from an embedded \texttt{scan} operation whose execution interval is enclosed within that of \( p_i \)’s \texttt{scan} operation.

If that embedded \texttt{scan} (by \( p_j \)) is direct, we are done. If not, we can apply the same argument inductively, noting that there can be at most \( n \) concurrent operations in the system. Hence, eventually the embedded \texttt{scan} is direct, and the result follows by the transitivity of the containment relation between embedded \texttt{scan} intervals.

We identify the following linearization points. A direct \texttt{scan} operation is linearized immediately after the read of the first collect in its successful double collect. An indirect \texttt{scan} operation is linearized at the same point as the direct \texttt{scan} whose view it borrows. Lemma 2 guarantees that such a direct \texttt{scan} exists and is entirely contained in the interval of the \texttt{scan} operation. Thus, all scan operations are linearized inside their intervals. An \texttt{update} operation by \( p_i \) is linearized when it writes to \texttt{Segment}[i].

**Theorem 3** The algorithm is a wait-free simulation of atomic snapshot object from read/write registers.

**Proof:** From Lemmas 1 and 2, every \texttt{scan} operation returns a view that is the actual value of the atomic snapshot object at the linearization point of the \texttt{scan}.

The data values returned by a scan operation are simultaneously held in all the segments at the linearization point of the operation. Therefore, each scan operation returns the value for the \( i \)’th segment written by the latest update operation by \( p_i \) that precedes it in the linearization. This completes the proof of linearizability.

We now prove wait-freedom. Each unsuccessful double collect by \( p_i \) can be attributed to some \( j \neq i \), for which the condition of line 5 does not hold. Thus, each such unsuccessful double collect increases \( b[j].ts - c[j].ts \) by 1. By the pigeonhole principle, in \( n \) unsuccessful double collects, 2
increase the difference $b[j].cs - c[j].ts$ for the same $j$. Thus, the `scan` procedure returns after at most $n$ double collects and the simulation is wait-free.