CSC2221: Introduction to Distributed Computing **Fall 2017**

Lecture 5: Linearizability

Lecturer: Faith Ellen *Scribe: Lily Li*

5.1 From Last Time...

An implementation of an object O is **linearizable** if for every admissible execution, there is a linearization π of the completed operation on O and some subset of the pending operations on O such that the response to each operation in the execution is the same as in π and if the response of the op occurs before the invocation of op' in the execution then op occurs before op' in π.

Theorem 5.1 *If an object of type T, shared by n processors, can be implemented (in a linearizable way — assumed) using only objects from type S and there is a problem P which can be solved by n processors using objects $S \cup \{T\}$. Then P can be solved in a system of n processors using objects of type S only.*

Definition 5.2 *(Alternative definition of π, above.) Each completed operation and some subset of the pending operations gets assigned a linearization point in its execution interval.*

Note: **wait-free** for n processors is equivalent to $(n-1)$-resilient.

Theorem 5.3 *There is no wait-free implementation of a Test&Set object shared by two processors using only registers.*

Proof: If there were a wait-free implementation of Test&Set object shared by two processors using only registers then there is a wait-free 2-consensus algorithm for two processors by Theorem 5.1. But this is impossible (by valency arguments from last week). ■

Theorem 5.4 *If the consensus number of X is m and consensus number of X' is $n > m$, there is no wait-free simulation of X' with X and read/write registers in a system with more than m processors.*

Proof: Suppose, for a contradiction, that there exists a simulation of X' using only X and read/write registers in a network with $k > m$ processors. Let $l = \min\{k, n\}$. Then there exists such a simulation in a network with l processors (if $l < k$ then just add $k - l$ dummy processors). Since the algorithm only uses X objects and read/write registers, the consensus number of X is $> l$. This is a contradiction. ■

5.2 Implementation of Useful Objects (From Registers)

5.2.1 Counter

The set of possible values is $\mathbb{N}$. The initial value is zero. Operation allowed are: $\mathsf{read}(R)$ (reads the value of R) and $\mathsf{inc}(R)$ (increments the value in R and returns an acknowledgement). See Algorithm 1.

Since each call to inc only increments the sum by 1, there exists a point between to invocations of inc which exactly corresponds to the expected value of any read call. Note that this implementation does not work when we are allowed to increment by arbitrary values. This is because the incrementation order may differ from the read order. Another possibility that you can think about is if both increment and decrement are allowed. Would this object be linearizable?

Algorithm 1 Counter Implementation from Registers.

```
inc($c$)
$y \leftarrow$ read($x_i$)
$x_i \leftarrow$ write($y + 1$) # Here is were need the linearization points.

read()
$y \leftarrow 0$
$y \leftarrow y + x_i$ for all $i = 0, ..., n$
return $y$
```

5.3 Large (L-bit) Single-writer Register

Note that the naive representation is not linearizable (why?). Instead we will use a tree with 2^L leaves. Further each internal node will store a bit (zero means left child and one means a right child); technically we use a heap instead of tree but that's just a technicality. read follows the internal nodes to a leaf and returns the value of the leaf. write(v) starts at the desired leaf and switches the nodes along its path to the root.

Wait! There is a problem. Can you see it? (Consider the following sequence of operations: write(2), write(1), and two reads, one which read a 1 and the second which reads a 2.)

Definition 5.5 *A register is* ***regular*** *if the value returned by each* read *is either the value written by the last* write *completed before the* read *began (or initial value if there were no writes) or the value written by a* write *operation concurrent with the* read*.*

Note that linearizable *is a stronger notion than* regular*.*

Claim 5.6 *This (the above) is a regular implementation of a register.*

Proof: Consider a read, R. There are two cases: (1) either R only reads nodes before they get written to or (2) R reads some node which has been written to. In case (1), R must read a 0 and further all writes must finish after R reads (do you see why?).

In case (2) let w be the last write such that one of the nodes v that R reads is written by w prior to R reading v. Why don't write operations over-lap? Two more cases! (a) Suppose that the child of v is some internal v' and that w wrote in v pointing to v'. Then w must have already written to v' (this is subtle, do you see why?). Continue this argument down the tree until we get to case (b), the child of v that w wrote in v is a leaf. Thus R return the value written by w.

Notice that we cannot have a write that occurs after w and before R began (why?). Thus R read the appropriate value as required. ■

So the above structure is not linearizable. Let us see what we can do to change that (hint: let us add more information into the tree). In particular we will now build a tree with $\binom{n}{2} \cdot 2^L$ leaves. read operations will be as before. write will change as follows: let v' be the old value and v be the new value. Find the node p in the second to last level such v' and v are the value of the children of p. We will perform *two* of our usual write operations. Suppose that the current node on the second to last level is p'. In the first write we will update the path from p''s v' to p's v'. In the second write we will update p's v' to p's v. Observe that this is an atomic operation. We will put our linearization point for the write operation here.

To show that this is correct suppose that R (that the read operation) returns the value a. If there is a write w of a linearized to within the execution interval of R, then simply linearize R to be sometime after the linearization point of w in the execution interval.

Otherwise we can argue that the write w which wrote a must have been the last write before the start of R and that all other writes have their linearization points after R has terminated.

5.4 Homework

Bounded counter using bounded registers (the size of the register can be some function of the bound).