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

Lecture 4: Consensus Hierarchy

Lecturer: Faith Ellen *Scribe: Lily Li*

4.1 Asynchronous Shared Memory Systems (4.1)

Assume that the system contains n processors, $p_0, ..., p_{n-1}$ and m registers $R_0, ..., R_{m-1}$. Each processor is modeled as a state machine, but there are no longer in-buffers and out-buffers (registers take the role of these). Registers haves a **type** which specifies:

1. Values that can be taken on by the register.
2. Operations that the register can perform.
3. Values that can be returned by each operation — possibly none.
4. New value of register resulting from each operation.

Further an initial value can be specified for each register.

Example 4.1 *Registers (aka read/write register): take values* $\mathbb{N}$ *or no value* $\perp$*, initial value is* 0 *(again or* $\perp$*) with operations:* $\mathsf{read}(R)$ *or* $\mathsf{write}(R, v)$ *(possibly* $R \leftarrow \mathsf{write}(v)$*).*

Single-write register where only the owner can perform a write operation.

A **step** consists of a processor, an object, and an operation. A **configuration** in the shared memory model is a vector

$$C = \langle q_0, ..., q_{n-1}, r_0, ..., r_{m-1} \rangle$$

where q_i is a state of processor p_i and r_j is the value of register R_j. Input event provides input to one processor and output event are performed by the processor and does not change the state of memory. Let $\mathsf{mem}(C)$ be the state of **memory** in C i.e. $\langle r_0, ..., r_{m-1} \rangle$. In the **initial configuration** all processors are in their initial states and all registers contain initial values. The **events** in a shared memory system are computation steps by the processor: these can be input, output or steps. An **execution fragment** is a finite/infinite alternating sequence of configurations and events:

$$\alpha_1 C_1 \alpha_2 C_2 \cdots$$

An **execution** is an execution fragment beginning at an initial configuration. If processes and objects are deterministic then execution is uniquely determined. If α is a finite sequence of events then $C\alpha$ is α applied to config C. At each step processor p_i can:

1.

C' is **reachable** from C if $\exists$ a finite sequence of events α such that $C' = C\alpha$. General C' is reachable if it is reachable from an initial config.

A **schedule** in the shared memory model is a sequence of processor indices indicating the sequence of events in an execution. A schedule is **P-only**, if the schedule consists only of indices for processors in P. If $P = \{p_i\}$ then you can write p_i-only.

Definition 4.2 *Configuration C is **similar** to configuration C' with respect to a set of processors P, denoted $C \overset{P}{\sim} C'$, if each processor in P has the same state in C as in C' and $\mathsf{mem}(C) = \mathsf{mem}(C')$.*

Lemma 4.3 *If $C \overset{P}{\sim} C'$ and α is a finite sequence of events by process in P, then $C\alpha \overset{P}{\sim} C'\alpha$.*

A process crashes in an infinite execution if it only performs a finite number of steps.

4.1.1 Complexity Measure

Focus on **space complexity**, that is the amount of shared memory needed to solve the problem. This is measured in two different ways: number of distinct shared variables and the amount of shared space (number of bits, number of distinct values). You could also consider the size of the objects (I guess that is pretty similar to the first one). You can still count the number of steps required to solved the problem but the major concern here is to see if it is finite, infinite, or bounded (worst-case number of steps a processor takes to solve a problem). Another metric is work: which is the worst case total number of steps taken by all the processors.

4.1.2 Consensus Algorithm

Suppose we have processors $p_1, ..., p_n$ and registers $r_1, ..., r_m$. Here are two different ways to solve the problem. If $m = n$ then we can have every processor write to its own register and read from all the other registers, taking minimum if

Example 4.4 *Test&Set Object: the possible values are $\{0, 1\}$. The initial value is 0. The available operations are Test&Set which returns the current value and flips it to 1 if it is currently zero and Reset which returns value to 0.*

Example 4.5 *Compare&Swap Object (CAS) set of possible values is $\mathbb{N} \cup \{\perp\}$. Initial value is 0 or $\perp$. Operation $CAS(R, u, v)$ check if $R = u$ then sets R to v returns the original value of R. How can you do a read if you only had CAS?*

A closely related object is Compare&Set (this is weaker than CAS but only returns $True$ or $False$ depending if the write was successful). Consider how to solve binary consensus.

How would you solve $(n-1)$-resilient consensus using only 3-valued CAS objects $\{\perp, 0, 1\}$ and binary registers. Note: values are not necessarily binary. (The process is actually quite involved! And there are actually two different ways to do this! One which requires $\log n$ CAS and the other which has $\log m$ CAS, where m is the length of the message.)

First we will solve $(n-1)$-resilient binary consensus. Then we will use this restricted form of the problem to bootstrap to arbitrary integer valued inputs. Here are the two different ways to achieve binary consensus: every processors writes its own value to a register. Then the processor competes to put its own value into the CAS object. If it wins, its done out put its own value. All other processors will try to compete for the value and get the value of the first processor. Output this value.

Now lets bootstrap: let the length of the longest value be $\log m$*. Now every processor gets* $\log m$ *binary registers to write their input values and need to compete on* $\log m$ *different binary CAS objects (one for each bit). As before a processor write all its input bits then starts competing starting from the most significant bit. Now observe that several things can occur: (1) there can be different winners for the different CAS objects and (2) you can read partially completed values off of some other processor's register since they have not yet finished writing. To address the first issue we make the following changes: each input register now has an additional* finished *bit. All processors that did not win a CAS will look at all the input registers which have been marked finished and match the first set of completed CAS objects. This processor will adopt this new value as their own and champion for it to be filled into all the CAS objects. Observe that such a register must exists since processors finish writing before they begin to compete.*

Here is the other way to solve this problem. Again we will start by solving binary consensus. This time we need n *CAS objects where* n *is the number of processors. We are going to put the CAS objects into the shape of a complete binary tree. We want to think of the the CAS objects now like competition brackets. Each processor stores a register in which it writes its input value. Now for each pair of processors* $\langle p_i, p_j \rangle$ *in a competition bracket the left processor tries to write a zero and the left processor tries to write a one. Similar to before, all losing processors must adopt the direction (*$\{0, 1\}$*) of the winner, just in-case the winner dies.*

4.2 Consensus Hierarchy (15.2)

(In the book this section is titled wait-free hierarchy.) Remember Sam talked about how consensus is a big deal. Object type X **solves wait-free** n**-processor consensus** if there exists an asynchronous consensus algorithm for n processors, up to $n-1$ of which might fail (by crashing), using only shared objects of type X and read/write objects.

Consensus number of object type X is n, denote $CN(X) = n$ if n is the largest value for which X solves wait-free n-processor consensus (aka there is an $(n-1)$-resilient consensus algorithm). If X solves wait-free n-processor consensus for every n then $CN(X) = \infty$. Note that $CN(X) > 1$ for all X (why? Because if you only had one processor then it could just output its own value).

Below are some examples of objects of different consensus numbers:

$\mathsf{cons}(X) = 1$: read/write register

$\mathsf{cons}(X) = 2$: FIFO queue, test&swap (Test&Set), swap, fetch&and, stacks

$\mathsf{cons}(X) = \infty$: compare&swap

For all positive integers m there exists objects with consensus number m.

Example 4.6 *Define type* Q_m *as follows. Let the values be binary strings of* $\leq m$ *and* $*$*. Operations are (1)* $APPEND(b)$ *which appends the bit* b *if your string is not too long otherwise turns into* $*$ *(*$APPEND$ *to* $*$ *gives you a* $*$*) and (2)* $FIRST$ *which returns the first bit of the string, if string is empty return* $\perp$*.*

We will show that the $\mathsf{cons}(Q_m) = m$*. First we show* $\mathsf{cons}(Q_m) \geq m$*. If we are doing binary consensus* $x_i \in \{0, 1\}$ *every calls* $APPEND(Q_m, x_i)$ *then returns* $FIRST(Q_m)$*. (Do you see why this works?)*

Next show $\mathsf{cons}(Q_m) < m+1$ *by a valency argument. We suppose that there exists an algorithm which solves binary consensus for* $m+1$ *processors using* Q_m *objects. Please finish the rest of the proof by a critical configuration argument. Remember, the structure is that you need to look at every operation and see which ones are not commutative. Then you want to look into those and show that there is still a problem.*

Theorem 4.7 *If $CN(X) = m$ and $CN(Y) = n > m$, then there is no wait-free simulation of Y with X and read/write registers in a system with more than m processors.*

Theorem 4.8 *There is no $(n-1)$-resilient binary consensus algorithm for $n \geq 2$ processes in an asynchronous shared memory system using only (read/write) registers.*

Proof: (Valency result similar to FLP) Suppose for a contradiction there exists such an algorithm. Recall that there exists an initial bivalent configuration. We will show that there is no reachable critical-configuration. Suppose C is a reachable configuration. Such that for step α_i for processor p_i we reach a univalent configuration C_i. Let C_i be 0-univalent and 1-univalent. Recall the commutativity result as before. If α_i and α_j access different registers or are read operations, then they commute and there is an issue. Next suppose that α_j is a write to register R (α_i could be a read or a write). Consider $C\alpha_j$ and $C_i\alpha_j$. In both cases the same value is in R and the state of p_j is the same so $C\alpha_j \overset{p_j}{\sim} C_i\alpha_j$. Contradiction (why?). ■

Corollary 4.9 *There is no $(n-1)$-resilient consensus algorithm for $n > 2$ processes using T&S and registers.*

Can you run through this algorithm for me?

Algorithm 1 Algorithm for BA in a complete network: code for processor p_i.

```
1: Broadcasts input value u_i
2: for rounds 2, ..., f + 1 do
3:     Broadcasts all the information it receives in that round
4: end for
5: Compute u_i output from all the messages it receives using something like a recursive majority function.
```

4.3 Implementations

(In preparation for Tarjan's talk.) We have some input and output events as before. An operation is pending if it is an input and output even that has not yet completed. Processors can only have one pending operation at a time.

Linearizability is the correctness condition we are going to enforce (basically every operation is turned into one atomic operation). An implementation of an object O is **linearizable** if for every admissible execution (every operation has ≤ 1 pending invocation at each configuration) has a linearization (aka ordering, permutation) π of the completed operations on O and some subset of the pending operations such that the response to each operation in the execution is the same as its sequence in π. Further the order of the responses must inform the order of π.

Basically you can put down a point for each execution (as it it were atomic) within its invocation and response such that the points are ordered.

4.4 Homework

Show that it is impossible to solve binary consensus using one Test&Set object. Show further that it *is* possible to solve binary consensus using two Test&Set objects. Is 1-resilient consensus for three processes using only Test&Set registers? (No, but I don't remember the proof.)