Reading: Ch. 13, 17
Next: 19, 21 (skim) 

*FD papers (Chadra & Toueg)

Last time: Defined atomic objects, showed that they can replace sh. vars + users can't tell the difference.
Then started giving algo for implementing atomic objects in particular, atomic snapshots.

Today: Implementing R/W atomic objects
And on to relating the sh. var model to the dist. network model.
Read/write atomic objects

\[ m \text{ write ports} \]

one register, \( m \) writers, \( p \) readers

\[ m = m + p \]

\[ m+1, \ldots, n \]

\[ \text{MMWR} \]

Atomicity for registers is a strong condition

Difficult to implement from weaker primitives like IWR regs

in distributed networks

Vitanyi-Awerbuch

From IWR regs

Use \( n^2 \) sh vars, in matrix:

\[
\begin{array}{cccc}
\text{WRITE} & \text{proc} & \text{write} & \text{READ} \\
\text{proc} & \text{read} & \text{proc read} & \text{READ proc read} \\
\end{array}
\]

\[
\begin{array}{cccc}
1, \ldots, m & m+1 & \ldots, n \\
\end{array}
\]

Variables have

\[
\begin{array}{ccc}
\text{val} & \text{tag} & \text{index} \\
\text{initially} & 0 & \text{any (but all the same)} \\
\end{array}
\]

\[ \text{valid sequence # } \leq \text{(disadvantage) } \]

\((\text{tag, index})\) pairs ordered lexicographically.
**WRITE** \((v)\) :: \(i\) reads all slots in its row

Let \(k\) = largest tag it sees.

Now \(w_i\) writes to all in its col:
\[
\begin{align*}
\text{val} & := v \\
\text{tag} & := k + 1 \\
\text{index} & := i
\end{align*}
\]

**READ** :: \(i\) reads all in row

Let \((v, k, j)\) be a triple with max \((\text{tag, index})\)

Propagate value in its column:
\[
\begin{align*}
\text{val} & := v \\
\text{tag} & := k \\
\text{index} & := j
\end{align*}
\]

Then output \(v\).

**Proof** (That this implements wait-free atomic R/W regs):

Well-formed, wait-free easy.

Show atomicity.

Could try to proceed as in snapshot proof, describing explicitly where to put the seq. pts.

But actually, it's not that easy to see where to put them

(Not just a single write step as in snapshot, e.g.)

Some kind of race condition involving write in col + others reading in their rows.

Turns out to be helpful to establish a partial order of ops, based on \((\text{tag, index})\) pairs, + prove the p.o. satisfies several conditions.

Q: What combs?

As in Lemma.
Lemma: Let \( \beta \) be a (finite or infinite) sequence of ops \( \phi \) reaps (for RW atomic object) that contains no incomplete ops.

Let \( \Pi \) be the set of ops in \( \beta \).

Suppose \( \alpha \) is (irreflexive) p.o. of all the ops in \( \Pi \), satisfying:

1. For any \( \pi \in \Pi \), there are only finitely many \( \phi \) such that \( \phi < \pi \).
2. If \( \phi \) precedes the \( \pi \) for \( \phi \) in \( \beta \), then it cannot be the case that \( \phi < \pi \).
   (consistent with external order)
3. If \( \Pi \) is a WRITE in \( \Pi \), \( \phi \) any op in \( \Pi \), then either \( \pi \Pi \phi \) or \( \phi \Pi \).
   (Ordens all writes; all reads wrt all writes.)
4. Value returned by each READ op is value written by last preceding WRITE op according to \( \alpha \).
   (or \( V_0 \) if there is no such WRITE).

Then \( \beta \) satisfies atomicity.

Actually, careful observer will note that Cond. 1. isn't needed - follows for the others. (2 \% all ops complete)

Students noticed this last time...

Pf. If we have all this, then we can insert \( \alpha \Pi \).

Rule: Insert \( \alpha \Pi \) just after the latest of the invocations for \( \Pi \) and all the ops \( \phi \) such that \( \phi < \Pi \).

(Cond. 1. implies well-defined.)

(Continuus \( \alpha \Pi \) is ordered consistently with \( < \).)
Since this works, first note that the order of the \( \ast \) is consistent
with <
\[
(\text{if } \phi < \tau \text{ then } \ast \prec \ast \tau)
\]

Show \( \phi \prec \tau \) in required intervals:
Clearly after invocation of \( \tau \).
But could it be also after the response?
Claim not. If so, then \( \tau \) has \( \phi \) as \( \ast \) with \( \phi < \tau \) comes
after \( \ast \ast \tau \).
Violates condition 2.

Show each READ returns value of WRITE whose \( \ast \) \( \tau \) comes right
before READ's \( \ast \).
(to show serial trace is produced)

Condition 3: copy all WRITEs ordered \( \ast \tau \) all others,
Cond 4 keep READ returns result written by last proc.
WRITE in the p.o.
Since the \( \ast \tau \) order is consistent with this p.o., that is the
needed value.

Use this to show correctness of V-A algorithm.
Take exe. 2 of V-A, assume no incomplete op.
Construct a p.o. w/ the 4 properties.

Based on the \((\text{tag, index})\) pairs:

Preliminary claim
Claim 1. \((\text{tag, index})\) values in any particular \(X(i, j)\) variable are
monotone nondecreasing in \(\tau\).

Pf. Always written by \(j\).
\(j\)'s \ast \tau\)s are sequential, \(\ast \tau\)s to entire column.
Each \(\ast \tau\) of \(j\) involves reading it now, then writing it of.
with tag pair \( \geq \) the max one it saw.
But saw its own \( \times (y, y) \) - same as \( \times (\ell, y) \) 
+ chooses something \( \geq \).

Claim 2: All tag pairs of distinct WRITES are \( \neq \)

\( \phi \): Different ports, different machines
Same port, sequential, see in drag, \( \psi \) > chooses larger

Now define p.o. + show 4 conditions

\( \psi < \phi \) if smaller tag pair

\( y = \) tag pair \( \psi \) WRITE \( \psi \) READ

Check 4 ends, two of them
I'll just do one \( \phi \) leave the rest to you.

Cond. 2: If resp for \( \psi \) precedes resp for \( \phi \) then can't have \( \psi < \phi \).

\( \phi \): Suppose

\[ \begin{bmatrix} \phi \\ \psi \end{bmatrix} \\ \text{resp. inv. for } \psi \text{ for } \phi \]

\( \psi \) has written its tag pair to its whole col.
So by Claim 1, \( \phi \) reads tag pair \( \geq \) that of \( \psi \)
+ chooses one that's at least as big
If WRITE, chooses strictly bigger.

This means ordered after, in order. can't have \( \psi < \phi \).

Cond. 3: WRITES ordered w.r.t. each other; \( \forall \) all READS

Because all WRITES get distinct tag pairs (Claim 2).
V-A algorithm not too costly in time but has unbounded wait
Implement atomic regs with bold tags
≡ several algo in literature \{ Burns, Peterson \}
\{ Lam-Shaun \}

Somewhat complicated, so LTIR.
Note: just do one special-case example
Cute, has min. relu proof

\[ x(1) \times x(2) \]

\[ x(1) \text{ writer by writer 1, read by all 3} \]
\[ x(2) \text{ " " " 2 " " "} \]
\[ x(c) \text{ holds } \begin{cases} \text{val} \in V, \text{init} V_0 \\ \text{tag} \in \{0,1,3\}, \text{init} 0 \end{cases} \]

\[ \text{WRITE}(V) : \begin{cases} \text{Read } x(c) & \text{ & get tag, say } t \\ \text{Write } x(c) : \begin{cases} \text{val} := V \\ \text{tag} := \text{tag} + t \mod 2 \end{cases} \\ \text{(Try to make sum of tags equal to } i \mod 2 \) \\ \text{(1 try to make tag \#}) \end{cases} \]

\[ \text{READ} : \text{Read both regs, let } b \text{ sum of tags, mod 2} \]
\[ \text{if } b = 1 \text{ read reg (1)} \]
\[ \text{else read reg (2)} \]
\[ \text{if return value found then} \]
Pf: well-paced, wait-free

Atomicity: don't use explicit sets:

- don't use p.c. buckets
- but, use bin mapping to an easier algorithm
  similar alg., but with integer tags instead of bits.

Now, $x'(i)$ holds:

\[
\begin{align*}
\text{tag} & \in \mathbb{N}, \text{ init } 50 \text{ for } i = 1 \\
& \textit{1 for } i = 2
\end{align*}
\]

Alg. WRITE(v):

- Read other to get integer tag $t$
- Write new value, with tag $t+1$

READ:

- Read both.

\[
\begin{align*}
|t_1 - t_2| \leq 1, \text{ read the one with the larger tag.} \\
\text{Else read either (non-deterministically)}
\end{align*}
\]

Q: Why this funny non-deterministic choice?

A: 1. Generality needed to make the sum work correctly

   - Also, doesn't really complicate the algorithm:
     1. Can equal correctness just as easily with this generality
     2. Removing unnecessary determinism makes it clearer why
        an algorithm is correct.

   Moral: Don't be afraid of non-determinism. A good thing.

\[
\begin{align*}
\text{Int:} & \quad \left\{ \begin{array}{ll}
x(1) \text{ tag always even} \\
x(2) \text{ odd}
\end{array} \right. \\
\text{Difference always} & = 1
\end{align*}
\]

Pf for integer alg.

- W-f; W-f easy

   Atomicity also easy, using p.o. lemma.
Define the p.c. by first ordering WRITEs by the tags they write, breaking ties (which must be by the same WRITE) in time order (sequentially).

I insert READ just after the WRITE whose value it gets.

Check the 4 conditions.

Eq. Cond. 2:

\[
\text{WRITE} \quad \text{READ}
\]

\[\text{tag } t\]

Suffices. Must show we can't have \(\phi < T\) which amounts to saying that the READ has to return either the result written by \(T\) or the value of some WRITE with a bigger tag.

When \(\phi\) invoked, \(\pi(i).\text{tag} \geq t\) by monotonicity.

At that instant, \(\pi(i).\text{tag} \geq t - 1\) by invariant.

So the only problem is if \(\phi\) returns value of some WRITE with \(\text{tag} = t - 1\). (Can't return anything less)

Suppose it does.

Then \(\phi\) must see \(\pi(i).\text{tag} \leq t - 1\) on first or 2nd read

Also on 3rd read

But if we see \(\pi(i).\text{tag} = t - 1\) on first or second read, then would reach \(\pi(i).\text{tag} = t\) conflict.

On the other hand, if \(\phi\) sees \(\pi(i).\text{tag} > t\), (the only other possibility)

Then by the time it sees this, \(\pi(i).\text{tag} > t - 1\).

So can't see \(t - 1\) on 3rd read, conflict.
Sim reln from Bloom to the integer alg.

Boolean tags are the 2nd low-order bits of the integers.
This is a multivalued sim reln, not an abstraction function.

\[
\begin{array}{c}
\text{Ex.} \\
\times (1) \\
0000 \\
0010 \\
0100 \\
\times (2) \\
0001 \\
0011 \\
0101 \\
e tc.
\end{array}
\]

\[
\begin{array}{c}
\text{Def.} \\
(s, u) = 1 \iff \text{all state components identical, except whenever} \\
u \text{has integer tag } t, s \text{has bit-valued tag = } 2 \text{nd low-order} \\
\text{bit of } u.
\end{array}
\]

Then: if is sim reln.


Step cond: Given \((s, u, s')\) in Bloom's alg., corresponding

step of \(s'\) is almost the same:

the same kind of action: \(\text{inv, resp, write, read1, read2, reads} 3\)

\& states fill in deterministically.

Having the state at step correct, must show enabling + preservation of

state correct.

Some key ideas (interesting cases):
1. Write step of WRITE preserves the correspondence.

Say by process 1:

Write \( x(1) \). Tag to be \( \pm \) value of \( \bar{b} \) it read from \( x(2) \). Tag in \( \textit{bloom} \) in \( \textit{Int} \).

Know by assumption about \( \textit{cache} \) or pre-states that \( \bar{b} = 2 \) not low-order bit of \( t \).

Want \( \bar{b} = t \).

What's written in \( \textit{bloom} \) the \( \textit{write} \) in \( \textit{bloom} \).

Ex.: \( B \quad t \quad \bar{b} \quad \bar{t} \)

0 101 1 110

Step 2nd low bit when increment an odd number.

It must be added because \( x(2) \) is always odd.

2. Enabling OK for 3rd read.

This says that \( \textit{int} \) allows reading the same tag specified in \( \textit{bloom} \).

Based on remembered tags read.

In \( \textit{bloom} \): \( \bar{b}_1 + \bar{t}_2 \) \( \bar{b}_1 = 2 \) nd low-order bit of \( t \),

In \( \textit{int} \): \( t_1 + t_2 \)

\( \bar{b}_2 = 2 \) nd low-order bit of \( t \).

Cases:

1. \( t_1 = t_2 + 1 \)

Then \( \textit{int} \) reads from \( x(1) \).

\( \textit{bloom} \) also, since \( t_1 \) even \( t_2 \) odd, so \( \bar{b}_1 \) and low-order bits to

\( \bar{b} \quad x(1) \quad x(2) \) unequal says read.

\( \bar{b} \quad x(1) \quad x(2) \) unequal odd

2. \( t_2 = t_1 + 1 \)

\( \textit{symmetric} \), both read \( x(2) \).
3. Neither
Then Bloom reads something but enables anything
yields trace inclusion, hence atomicity of Bloom.

This little toy alg. doesn't extend to 3 or more writers.
Algs that do are harder.
So more test, weaker guarantees than atomicity often made
(weaker coherence in multiprocessor memories)
But harder to program with non-atomic shared memory.