Reading: 12.2, 12.3, 13.1-3
Next: 13.4, 17
Then: 19, 21, 22...

Last tie: Began studying consensus in asynchronous model, in presence of stopping failures.
Impossible if even 1 failure.

Defined problem: Ascrib. like m.o.
init(v); decide(v);
well-founded: Only output 1 decision, v only if init previously occurred.
agreement: All dec values identical
validity: If init acts that occur all contain same v then that v is the only possible decision value

termination:

f-f termination: In any fair f-f exec in which init events occur on all ports, a decide occurs on each port.

Basic problem reps: w-f, agreement, validity, f-f-termination

fault-tolerance:

wait-free termination: In any fair f-f exec in which init events occur on all ports, a decide event occurs on every port

f-failure termination: 0 ≤ f ≤ n
In any fair exec in which init events occur on all ports, if there are stop events at most f ports fail, then a decide occurs on every non-stopping port

1-failure termination: The interesting case we consider
Impo ssibility of agreement (FLP)

Show there is no alg in R/W/sh. mem model that solves agreement with 1-failure termination.

Show in two stages:
1. Can't solve with wait-free termination.
2. Can't solve with 1-failure termination.

Restrictions: (WLOG)

V = \{0, 1\}

each alg is deterministic

\{ unique init state from any state, any process has \leq 1 l.c. action enabled from any state, for any action, exactly one new state \}

Every non-failed process always has a l.c. action enabled (even after it decides).

Terminology:
Initialization: Sequence of n init steps, 1 per port, in order order.
Input-first exec: Begins with initialization.

finite exec is 0-valid if 0 is the only value appearing in (this exec or) any extension; 0 actually does appear in some extension.

1-valid

invalid \triangleq 0-valid or 1-valid

leval \triangleq each of 0, 1 appears in some extension.

In absence of failures, classification is exhaustive:

Lemma: Each finite 1-val exec of A is either invalid or leval.
Let \( A \) be an alg. solvng agreement with 1-failure termination (on wait-free, which is strnger).

**Bivalent initialization**

**Lemma**: \( A \) has a bivalent initialization.

Says final decision can't always be determined just from inputs.

**Contrast** non-fault-tolerant case: can determine (e.g., take majority)

**pf**: Suppose not - then all initializations are equivalent.

\[ \text{init } x^0 = \text{ all 0's, is 0-valent?} \]

\[ \text{init } x^- = \text{ all 1's, is 1-valent?} \]

Construct chain of initializations, spanning from \( x^0 \) to \( x^- \).

Each consecutive pair differ in input of one process.

Must be 2 consecutive inits in chain, say \( x_i \) and \( x_i' \)

Says, they differ in initial value of \( i \).

Now consider fair case extending \( x^0 \), where \( i \) fails right after \( x_i \).

All but \( i \) must eventually decide, by 1-failure termination,

+ since \( x^0 \) is 0-valent, \( \text{dec } = 0 \)

\[ \text{Let } L \]

\[ \text{stop i} \]

all but i

\[ 0 \]

Extend \( L \) in same way, still get decision of 0.

Contradicts \( x^- \) 1-valent.
Now we can show the first main result - impossibility for wait-free termination.

Suppose $A$ has w-f term, get control.

Based on pinpointing how a decision must be made.

Def: Decider execution $x$ is finite, FF, input-first exec.

s.t.
1. $x$ is bivalent,
2. For every $i$, $\text{exit}(x, i)$ is univalent.

Define

Lemma 1: $A$ (with w-f term) has a decider exec.

Pf: Suppose not. Then any bivalent FF (input-first) exec. has a 1-step bivalent FF extension.

Then can start with bivalent init (existence shown above), produce infinite FF exec $\omega$ all of whose prefixes are bivalent.

So no one ever decides.

Now: At each stage, start with bivalent FF (input-first) exec. + extend by 1 step to bivalent (by assumption, possible).

$\omega$ must contain $\omega$ may steps of some process, say $i$.

Claim $i$ must decide in $\omega$, which is a contradiction.

Just add step events for all processes that take only fin many steps.

Renet is a fair execution.

Wait-free termination says $i$ must decide.

Lemma 2: $A$ can't exist (satisfying w-f termination, that is)

Pf: Fix $x$ decides $\omega$. 
Bivalent, so \( \exists \) 2 procs \( i \) \& \( j \), leading to
\( 0 \)-val \& \( 1 \)-val states, resp.

Case analysis yields contradiction:

1. \( i \)'s step is a read
   Then run all but \( i \) after exit \((x,y)\)

   \[
   \text{all but } i \text{ take } \omega \text{ many steps.}
   \]
   \[
   \text{Looks like a fair execution in which } i \text{ fails.}
   \]
   \[
   \text{So all others must decide, } + \text{ since } 1 \text{-valent,}
   \]
   \[
   \text{decide } 1.
   \]
   \[
   \\text{(start with } i \text{)}
   \]
   \[
   \text{Run all after } 2 i, \text{ do same, decide } 1.
   \]

   Contradicts \( 0 \)-valence.

2. \( j \)'s step is a read
   Symmetric.

3. \( i \)'s + \( j \)'s steps both writes, to different vars

   Then independent, consider both steps, in
   either order.
   Same resulting state.

   \[
   \text{Impossible - if we run all, no}
   \]
   \[
   \text{failures, required to decide, but}
   \]
   \[
   \text{neither decision is possible.}
   \]

   \[
   \text{descendant of } 0 \text{-val} + 1 \text{-val.}
   \]
4. Both writes, to same var

Then run all but $i$ after $x,y$ (as in case 1)
Eventually decide 1.
Run same after $x,i$, start with $j$.

all but $i$ looks like all-but-$i$ because $j$'s step
overrides $i$'s step.

Control $O$-value.

; This proves the first impossibility result, for wait-free termination.

Impossibility for 1-failure termination

Previous proof doesn't give this - discuss why:
In Lemma 2 proof, W-f termination condition used to say i
must decide in a fair exec. in which it doesn't fail.
So now assume only that A satisfies 1-failure termination.

Lemma 3: If $x$ is bivalent if input-first execution, i any
process, then $\exists$ ff-extension $x'$ of $x$ such that
ext$(x',i)$ is bivalent.
Suppose we have this lemma + finish impossibility proof:

Construct pair $tt$ (input-first) exec in which no process ever
decides, contradicting $tt$ termination.

Begin with bivalent initialization.

Then repeatedly extend, including at least one step of a
particular process at each stage.

(1 at first stage, 2 at second, ..., round roten)

Do this maintaining bivalence + avoiding failures - Lemma 3
implies each step possible.

Remains to prove Lemma 3:

Proof by contradiction: Suppose false, then must be some bivalent $tt$
(input-first) execution $2$ of $A$ and some $i$, such that for all
$tt$ extensions $2'$ of $2$, $\text{ext}(2', i)$ bivalent.

In particular, $\text{ext}(A, i)$ bivalent, WLOG $0$-valent.

Since $A$ bivalent, $z$ extension of $A$

deciding $B$, WLOG $tt$.

Consider running $i$ everywhere along the
"spine" of $tt$.

0 at beginning, 1 at end.

So $\exists$ 2 consecutive points:
Claim $j \neq i$

1. $i$'s step is a read

2. $j$'s step is read

Similar: Now consider states after $i$ ad $j$.

So can run all but $i$ in the same way. After both, must decide something, same in both, control $0$-val vs $1$-val.

states here indist. to all but $i$ (since read step of $i$)

but deterministic, so this isn't possible (descendant of $0$-val) can't be $1$-val.

Case analysis: (like that for the W-F proof)
3. Both routes, different way
Commulative scenario, get same control as before.

4. Both routes, same way

\[ \text{induct to all but } j \text{ (since } i \text{ precedes } j) \]
\[ \text{run all but } j, \text{ get control.} \]

This result is regarded as fundamental.
Also holds in asynchronous networks - will revisit shortly.
That's how it was originally proved.

Discuss impact - systems claims - have to say how they avoid this limitation.

\{ theory - spawned enormous amount of work trying to characterize what cannot be computed in asynchronous systems, in presence of failures. Algebraic topology methods used for characterization. \}
Importance of the data type (R/W)
Result doesn't hold for some more powerful data types

Ex: R-M-W sh. mem. Easy

1 sh var
Each accesses var, initially "unknown".
If see \( \text{(unknown)} \), then change to your own init value + decide that some actual value, accept it + decide that

New topic, still in area of F-T sh. memory algorithms:
atomic object

Atomic object = a version of a shared variable, but with invocations + responses not combined into one atomic step. Also, includes possibility of failures.

Provides useful building block for fault-tolerant distributed systems, or multiprocessor systems

Separating inc. + response allow for consideration of lower-level impls of these objects.
(There can use shared memory or distributed network impls)

Atomic object of a given type is like an ordinary sh var of that type, but also allows concurrent access.
Still looks "as if" ops occur one at a time, sequentially - in some order consistent with order of invocations + responses = linearizable objects
Also, fault-tolerance conditions

\[ \text{wait-free termination} \ \\
\{ \text{f-failure termination, etc.} \} \]

(stopping failures only)

Blocks for multiprocessor systems + distributed systems

Sometimes settle for

components with weaker coherence conditions.

Basic def:

\[ \text{Variable type} = (V, v_0, \text{invocations, responses, } f) \]

\[ f : \text{vars} \times V \rightarrow \text{resp} \times V \]

Describes response to an invocation (it change to var).

Can talk about execution of such a var. type:

\[ v_0; a_1; b_1; v_1; a_2; b_2; v_2; \ldots \]

\[ \text{value} \rightarrow \text{inv} \rightarrow \text{resp} \rightarrow \text{new value} \]

Trace is just the a's and b's.

Atomic object of a given type is E/O automation satisfying some

\{ well-formedness \}

\{ atomicity \}

\{ liveness \}

External interface:

Some subset of the invocations can be done on each port.

Not necessarily all everywhere.

Bs are corresponding responses.

Step 6 inputs, as before.
Compose with users \( U \), well-famed (alternate invs + responses)

Conditions on \( A \):
1. Preserves well-famedness (alt. inv + resp)
2. Atomicity

Describe when (w-f) sequence of inv + response actions is atomic.

Then \( A \) satisfies atomicity if all w-famed invs \( A \times U \) for any users have atomic traces.

So, say what it means for well-famed seq. \( \beta \) of invs + responses to be atomic.

If all invs had matching response (complete), would just say it's possible to insert a dummy action (serialization point) somewhere between each inv. + matching response, such that if the invs + responses were moved there (shrink down), then the result is a trace of the basic (serial) variable type.

Ex:

\[
\begin{array}{cccc}
\text{read} & \text{write(3)} & \text{OK} & 2 \\
\text{initial} & & & \\
\end{array}
\]

Atomic once can have

\[
\begin{array}{cccc}
\text{read} & \text{0} & \text{write(3)} & \text{OK} & 2 \\
\end{array}
\]

Ex:

\[
\begin{array}{cccc}
\text{read} & \text{write(3)} & 3 & \text{OK} & 2 \\
\text{read} & & & & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{write(3)} & \text{OK} & 2 & \text{read} & 1 \\
\end{array}
\]
Complication is incomplete operations

Can't require including them (since might not have gotten that far)
Can't require excluding them (since might have done all but return)
So leave it nondeterministic.

Require that it is possible to

1. Insert serialization gets for all complete ops (between inv response)
2. Select subset of incomplete ops (arbitrary)
   3. For each op in #2, select a response (make it up)

   Insert a ser pt somewhere after inv
   Such that shrinking (incl. new responses) leads to trace of basic
data type (must remove other invocations).

Examples:

```
                        incomplete, still ok
    0  read 1  write (3) 2  3 1
```

Ex:

```
                        incomplete, never gets ser pt, still ok
    0  write (3)  read 1  0  read 1  0  read 1  0  read ...
```

Some examples of non-atomic seqs:

Ex:

```
    write (3)  ok  read 0
Write not established
```

Ex:

```
    write (3)  read 3  read 0
Out of order
```