Consensus:

Impossible
or
Not?

Admin. Midterm Next Week

- During class.
- Open book. Feel free to bring papers, notes, etc.
- Focus on testing understanding of the papers & concepts covered in class:
  (please make sure you know what the asynchronous model entails)
- Questions in the pre-class notes are good examples of what might be asked

Recap

Want to build a fault tolerant application
Is A State Machine

Alice → OPA
Beth → OPR
Carolyn → OK

Requirement

→ All Logs Agree On Order Of Operations

All Logs Agree On Operation At
Idx 0, 1, 2, ...

→ Consensus Algorithms

Core Result

No Deterministic Fault-Tolerant Consensus Protocol
For the Asynchronous Setting

0. Deterministic
   What?

   Where used?

2. Fault Tolerant
   - Want to consider as many protocols as possible
     - Choose a weak failure model
       - At most 1 process fails
       - Fail-stop failures

3. Consensus Protocol

\[
\begin{array}{cccc}
\text{In}_0 & \text{In}_1 & \text{In}_2 & \text{In}_3 \\
0 & 1 & 2 & 3 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{P} & \\
\text{In}_0 & \text{In}_1 & \text{In}_2 & \text{In}_3 \\
\end{array}
\]
Requirements

**Agreement**: If Process \( P_i \) decides \( x_i \) and \( P_j \) decides \( x_j \), then \( x_i = x_j \).

**Validity**: If Process \( P_i \) decides \( x_i \), then there exists a \( t \) s.t. \( I_{x_i}^t = x_i \).

(Equivalent to Strong Unanimity)

**Termination**: All processes eventually decide.

Terminology

**Configuration**: State of all processes

**Event/Step**: \( \text{receive}(p, m) \)

Receive message \( m \) at process \( p \)

\( m \) can be \( \emptyset \), the empty message.

Changes the state of process \( p \).

\[ c = \text{receive}(p, m) \]

Must be enabled in \( C_0 \).
Schedule ($\sigma$): Sequence of steps.

$e_0$ disjoint from $e_1$: $e_0$ is a receive from a different process than $e_1$.

$\sigma_0$ disjoint from $\sigma_1$.

No deterministic fault-tolerant consensus protocol for the asynchronous setting.

Overall idea:

1. Focus on binary consensus [inputs & outputs are 0 or 1]

   Assume a deterministic, fault-tolerant consensus protocol exists $P$.

2. Show that $\exists$ configurations for which $P$ can decide both 0 or 1 [bivalent configs]

3. Show that $\exists$ initial configurations that are bivalent
Show that one can always find a schedule that goes from bivalent C to another bivalent C

"Construct a schedule"

Why does this work?

P is 1-fault tolerant

But works in async networks

\[ \text{Cannot distinguish b/w delayed messages \& failed process} \]

Must eventually suspect quiet processes have failed
Also must deal with quiet processes sending messages

**Diamond Lemma**

\[
P_i = \emptyset, \quad P_2 = 2
\]

\[e_1, e_2 \text{ disjoint } \Rightarrow e_1 = \text{new}(P_i, n), \quad e_2 = \text{new}(P_2, n)
\]

\[e_1, e_2 \text{ enabled in } C \quad P_i \not\models P_2
\]

\[
\begin{array}{c}
\sigma_1, \sigma_2 \text{ disjoint} \\
\text{enabled in } C
\end{array}
\]

\[
\begin{array}{c}
0000 \\
0000 \\
\end{array}
\]

**Valence**

Choose 0 / Decide 0
Bivalent Initial Configuration

Input | Output
0 0 0 0 | 0
0 0 0 1 | 0
0 0 1 0 | 0
0 0 1 1 | 0
0 1 0 0 | 0
0 1 0 1 | 0
0 1 1 0 | 0
0 1 1 1 | 0
1 0 0 0 | 1
1 0 0 1 | 1
1 0 1 0 | 1
1 0 1 1 | 1
1 1 0 0 | 1
1 1 0 1 | 1
1 1 1 0 | 1
1 1 1 1 | 1

Observation

P is 1-Fault tolerant

\[
\begin{array}{c|c|c}
0 0 0 & 0 1 0 & 0 0 1 \\
0 1 0 & 1 1 0 & 0 1 1 \\
1 0 0 & 0 0 1 & 1 1 1 \\
1 1 0 & 1 1 1 & 1 1 1 \\
\end{array}
\]

Choose 1 | Decide 1
Must decide even if \( \sigma \) where a process takes no step.

**Aside: Fairness?**

1. Assume \( P \)
2. \( \exists \) initial bivalent config
3. Show \( \exists \sigma \) going from bivalent config to bivalent config
   - Current bivalent config \( C_0 \)
   - Pick enabled event \( e \)
     - \( e(C_0) \) is bivalent \( \Rightarrow \) done
     - \( e(C_0) \) is not bivalent
       - \( \xrightarrow{\text{Find enabled } \sigma \text{ s.t.}} \)
       - \( C_0 \xrightarrow{\sigma} C_1 \xrightarrow{e} C_e \xrightarrow{\top} \) bivalent

Finding is hard (not really — constructive proof of FLP)

Instead show existence:

- Given \( C \) bivalent, enabled \( e \), \( \exists \) enabled \( \sigma \)
Proof by contradiction

No such $\sigma_0$ exists

All $D_i$ are 0-valent or 1-valent

What does $D$ contain?
Must contain at least one 0-valent & one 1-valent configuration.

Why

$C$ is bivalent

$\exists$ embedded $\sigma_0$ s.t.

$C \xrightarrow{\sigma_0} C_0 \Leftarrow 0$-valent

If $e \notin \sigma_0$ then $C_0 \xrightarrow{e} e(C_0) \notin D$

$0$-valent

If $e \in \sigma_0$ then

$\sigma_0 = e \odot _e e_i \odot _i e_j \odot j \ldots$

Imagine same thing with
contains both 0- & 1-valent configurations

\( \mathcal{C}_e \) contains configurations \( C_A, C_B \) that are

- Neighbors \( C_A \xrightarrow{f} C_B \)

- \( e(C_A) \) is 0-valent, \( e(C_B) \) is 1-valent.

How? Hand-wavy but use initial bivalent argument.

Finally: arrive at contradiction: no such \( f \) can exist

- If \( f \) and \( g \) are different processes
2. $e$ and $\sigma$ are not disjoint

$e, f \models p$

IP formulae

$\sigma \models (\exists e)\neg e$

Must be 0-decided

$e \models$ some schedule disjoint from $e$

$\sigma \models$ leads to a decision

$\Rightarrow \sigma$ disjoint from $\sigma$

$\Rightarrow \sigma$ disjoint from $f$

CONTRADICTION

Found enabled $\sigma$ s.t.

$C_0 \overset{\sigma}{\rightarrow} C_1 \overset{e}{\rightarrow} C_0 \overset{\uparrow}{\rightarrow}$

Bivalent
No \underline{Deterministic} Fault-Tolerant Consensus Protocol for the Asynchronous Setting

What happens if we relax assumptions?

1. \underline{Deterministic}?

2. \underline{Partial Synchrony}