FLP+ Partial Synchrony

Where we are

RSMs

\[ \Rightarrow \text{Consensus} \]

\[ \Rightarrow \text{Hard because FLP Impossibility} \]

\[ \Rightarrow \text{Circumvent by assuming Partial Synchrony} \Rightarrow \text{Raft} \]

\[ \Rightarrow \text{Circumvent with Randomness} \Rightarrow \text{Rabia} \]

Today 0: FLP, Partial Synchrony.

Defining Consensus

\[ i_A \] \[ i_B \] \[ i_C \]

\[ A \] \[ B \] \[ C \]

\[ A \] \[ B \] \[ C \]

\[ \text{decide (D_B)} \]

\[ A \] \[ B \] \[ C \]

\[ \text{decide (D_C)} \]

Agreement

\[ \Rightarrow \text{Process } p_0 \text{ decides } d_0 \& \text{ process } p_i \text{ decides } d_i \]

\[ \Rightarrow d_0 = d_i \]

Validity

\[ \Rightarrow \text{Process } p \text{ decides } d \]

\[ \Rightarrow \text{some process } q \text{ had input } = d \]
Impossibility Result:

No fault-tolerant, deterministic algorithm in the asynchronous model.

Aside: From an award speech in 2001

- Lynch wanted to prove an asynch consensus protocol (designed by Lamport) correct in '82.
- Kept running into trouble, started working with Fischer & Paterson. No luck
- Led to argument in this paper.

Going to walk through proof a bit

Why? - Ideas are interesting & at the core of how we think of many distributed algorithms
- Origin of the model we have been using.

How? Assume a simple setting
- Binary Consensus
- No message loss (only delays)
- At most 1 failure

System Model

Event \( e^o \) \( \text{recv} (p, m) \)

Can differ in state of process \( p \) \& set of pending messages

Schedule \( \sigma = e_1; e_2; e_3; \ldots; e_n \)

\( ? \) Valid \( \sigma \)?
"Partially" Correctness (on safe correctness)
  - Agreement
  - Validity

"Totally" Correct = partial correct + Liveness
  Termination

No fault-tolerant, deterministic algorithm in the asynchronous model.

Some tools

(1) \[ C \{ e_1, e_2 \} \]

1. \( e_1, e_2 \) enabled in \( C \)

\[ e_1 = \text{recv}(p_1, m) \quad e_2 = \text{recv}(p_2, m') \]

\( p_1 \neq p_2 \)

Diamond Lemma
2. Valence: Assume a partially correct consensus protocol \( \text{P} \) implemented by system.

\[
C \quad \text{\( \emptyset \)-valent iff \( \emptyset \) valid schedules from } C
\]

\[
\begin{array}{c}
\xrightarrow{\sigma} \\
\emptyset \text{-decided} \\
\text{(some process } p \text{ decided } \emptyset) \\
\text{OR} \\
\text{Undecided}
\end{array}
\]

Similarly \( 1 \)-valent.

\[
C \quad \text{bivalent } \Rightarrow \quad C \text{ is not } \emptyset \text{-valent } \land \\
C \text{ is not } 1 \text{-valent } \land \\
\text{some configuration reachable from } C \text{ by an enabled schedule decides}
\]

F/P proof \( \Rightarrow \) Proof by contradiction

Assume \( \exists \) totally correct protocol \( \text{P} \) that is \( 1 \)-fault tolerant.

\( 0 \in \emptyset \)- and \( 1 \)-valent initial configurations
2. \exists \text{ Bivalent initial configuration}

3. For any enabled event $e$ in bivalent configuration $C$, can find valid $\sigma$ (where at most 1 process is silent) s.t.

\[
\begin{array}{c}
C \\
\sigma \\
e \\
C'
\end{array}
\]

$C'$ is Bivalent.

4. Start from initial bivalent configuration
Schedule of unbounded length going from one Bivalent Config to another
⇒ Unbounded schedule where no process decides
⇒ No termination.

Look at (3)

Given bivalent configuration \( C \), event \( e \) want to show \( \exists \sigma \) s.t. \( C \xrightarrow{\sigma} e \) Bivalent

- If \( C \xrightarrow{e} \) bivalent : Done \( \sigma = [3] \)
- Otherwise,
Observation: Claim is equivalent to claim that \( D \) contains bivalent configuration.

Proof by contradiction:

(i) \( \exists \) reachable \( \phi \) \& 1-valent configuration from \( C \)

[Reachable: \( \exists \) valid \( \sigma \) from \( C \) to config]

- \( \forall \) totally connect \( \Rightarrow \) terminates
- \( C \) bivalent \( \Rightarrow \exists \sigma_0, \sigma_1 \)

\[
C \xrightarrow{\sigma_0} \text{decide } \phi \\
\xrightarrow{\sigma_1} \text{decide } 1
\]
(ii) \( \exists \emptyset \neq \emptyset \)-valent configurations in \( D \)

let \( c_0 \) be \( \emptyset \)valent config reachable from \( C \)
- If \( c_0 \in R_e \), then 
  \[ e(c_0) \in D \text{ is a } \emptyset \text{-valent config.} \]
- If \( c_0 \notin R_e \), the
  \[ e \in \text{ schedule } c_0 \text{ from } c_0 \xrightarrow{\sigma} 6 \]

\[ \Rightarrow \exists C_1 \text{ s.t. } \]
\[ C \xrightarrow{\sigma_0 \{ \text{before } e \}} C' \xrightarrow{e} C_0 \xrightarrow{\sigma_0 \{ \text{after } e \}} C_0 \]

\[ \Delta \ni c_0' \in D \]
But by assumption no bivalent conf in \( D \) \( \Rightarrow \) \( c_0' \) is \( \emptyset \)-valent

By symmetry for 1-valent

(iii) Hand-wavy:

\[ \exists c_0', c_1 \in R_e \text{ s.t.} \]
\[ \exists e' \quad c_0 \xrightarrow{e'} c_1 \]
\[ e \circ \sigma \quad \text{or} \quad \exists \sigma \text{ where } C_0 \xrightarrow{\sigma} \text{ some process decides} \]
\[ \sigma \text{ finite or } p \text{ takes no steps in } \sigma \]
[Because F.T.]
Thus D must contain a bivalent config.

Given bivalent configuration \( C \), event \( e \)

\[ \exists \sigma \text{ s.t. } C \overset{e}{\rightarrow} \text{Bivalent} \]

\[ \rightarrow \text{For any consensus protocol that is } 1\text{-FT, } \exists \text{ unbounded schedule w/o decision} \]

\[ \Rightarrow \text{No totally correct protocol.} \]

So where does that leave us

\[ \frac{\text{FT- consensus}}{\text{synchronous}} \left( \frac{\text{Real world??}}{} \right) \frac{\text{No consensus}}{\text{asynch}} \]
Partial Synchrony

- Separates out comm. & processing

Talks about two variants

(a) Known

Bounds don't hold

(b) Unknown

Bounds hold

Results are the same, but most people mean (a)

Approach

- Design protocols for a model where communication & processing are synchronous
- Messages can be lost
  ...
  But 3 time (GST) after which message from correct process p to correct process q is received

- Show they are totally correct

→ Design

- Rounds where processes send & receive messages

- Decisions require quorum of \textit{correct} processes to be involved

Why termination

- Each round takes bounded time
  (synchrony assumption)

- After GST, all correct processes can communicate w/ each other

  Assume large enough set of processes to ensure $k$-failures
  still allow decisions

→ Bounded rounds to reach consensus
→ bounded time to reach consensus.

- Show that synchronous rounds can be emulated in partially synchronous model

  → When $\Delta$ is known, use that to decide how long to wait for communication

  → When $\Delta$ is unknown, keep increasing until it hits the right value.