The consensus problem, slightly generalised

- Decision value is binary
  - Output register initially $\perp$, can only be written once.
- Consistency
  - all correct processes decide on the same value
- Non-triviality
  - there is an execution in which 0 is decided, and there is an execution in which 1 is decided
    - Note that we do not assume any relationship with the input values.
- Input combinations
  - All combinations of per-process allowed input values constitute a possible starting configuration
    - To disallow problems where the output is already ‘baked’ into all possible inputs
- Asynchronous

The consensus problem, slightly generalised

- 1-crash run: at least $n - 1$ processors take infinite number of steps
  - A terminating processor can always be augmented with a busy while loop
- 1-crash fair run: a 1-crash run where all correct processors receive messages sent to them
- Termination: In every 1-crash fair run all correct processors decide
Fischer – Lynch – Paterson impossibility

- No distributed protocol exists with these five properties: consensus, non-triviality, input-combinations, asynchronous, termination.

- Paraphrased: consensus cannot be solved in a completely asynchronous system.

Preliminaries (1)

- Recall configuration $C$ contains local state $C[p]$ of processor $p$ as well as state (i.e., buffer of messages in transit) $C[e]$, of edge $e = (p,q)$.

- Events, i.e., processor steps, move the system from one configuration to the next.
  - A step is a pair $(p,m)$ that is applicable in a certain state of processor $p$ when $m$ is in transit to $p$; it changes the state of $p$ and may send a new message. If $m = \emptyset$ then the step is applicable independent of messages being in transit.

- Let run $\sigma = a_1, a_2, a_3, ..., a_n$ then $\sigma(C)$ is the configuration $C'$ that results from applying the steps $a_1, a_2, a_3, .., a_n$ to $C$ in sequence.
  - A configuration $C$ is reachable from $C$ if there is a finite run $\sigma$ such that $C' = \sigma(C)$.
  - We write $\Sigma(C)$ for the set of all configurations reachable from $C$.

Preliminaries (2)

- A configuration $C$ is accessible if it is reachable from some initial configuration $C_0$.

- A configuration $C$ has decision value $v$ if $C[p].decision = v$ for some processor $p$.
  - A configuration is $v$-valent if $v$ is the only decision value for all reachable configurations.
  - A configuration is bivalent if both decision values are reachable.

- A run is admissible if at most 1 processor is faulty (and all messages to non-faulty processors are eventually received).
  - A processor is non-faulty if it takes infinitely many steps.
Let $P$ be a consensus protocol

$P$ is partially correct if

- No accessible configuration has more than 1 decision value
- For $v \in \{0, 1\}$ there is an accessible configuration with decision value $v$

$P$ is deciding if some processor decides in that run

A run is deciding if some processor decides in that run

$P$ is a totally correct consensus protocol in spite of one fault if it is partially correct and all its admissible runs are deciding

Idea:
- show that every partially correct protocol for the consensus problem has some admissible run that is not deciding.

Lemma 1: commutativity of steps

Lemma 1: If runs $\sigma, \sigma'$ are disjoint (i.e. no processor takes that takes a step in $\sigma$ occurs in $\sigma'$ and vice versa), then they can be applied in either order to reach the same final configuration. i.e. $\sigma(\sigma'(C)) = \sigma'(\sigma(C))$.

Proof of lemma 1

Steps on one processor never disable steps on other processors (they can only send a message and hence enable an action.)
Lemma 2: bivalence exists initially

- **Lemma 2**: It has bivalent initial configurations
  - Suppose not. Then by partial correctness, \( P \) has both 0 and 1 valent initial configurations.
  - Let \( C_0 \) be 0 valent and \( C_1 \) be 1 valent, such that they differ only on the initial state of some processor \( p \).
  - Start in some 0-valent configuration and change the state of one processor at a time to its state in some fixed 1-valent configuration. At some point you cross from 0-valent to 1-valent. These are the \( C_0 \) and \( C_1 \) we need.
  - Let \( e \) be an admissible deciding run starting in \( C_0 \) in which (only) \( p \) takes no steps. (Such \( e \) exists because the protocol is partially correct, and it is admissible because only \( p \) takes no steps.) Let \( v \) be the decision value.
  - Let \( \sigma \) be an admissible deciding run starting in \( C_0 \) in which (only) \( p \) takes no steps. (Such \( \sigma \) exists because the protocol is partially correct, and it is admissible because only \( p \) takes no steps.) Let \( v \) be the decision value.
  - Start \( \sigma \) in \( C_1 \). Because \( C_0 \) and \( C_1 \) are equivalent except for the initial state of \( p \), and because \( p \) does not take any steps in \( \sigma \), the resulting configuration is equivalent except for the state of \( p \). Again the decision value is \( v \).
  - But then, if \( v = 1 \), \( C_0 \) is bivalent while if \( v = 0 \), \( C_1 \) is bivalent.

Lemma 3: bivalence cannot be avoided

- **Lemma 3**: Let \( C \) be bivalent and let \( a \) be an event enabled for \( p \) in \( C \).
  - Let \( X \) be the set of configurations reachable from \( C \) without applying \( a \), and let \( Y = \{ a(C') | C' \in X \} \).
  - Note that \( p \) may execute other events!
  - Because \( a \) is enabled in \( C \) it is enabled in all \( C' \in X \).
  - Then \( Y \) contains a bivalent configuration.

Proof of lemma 3

- Suppose \( Y \) contains no bivalent configurations
  - Let \( E \) be a \( 0 \)-valent configuration reachable from \( C \).
    - Both exist because \( C \) is bivalent.
  - If \( E \in X \) let \( F = a(E) \in Y \); otherwise \( a \) was applied to reach \( E \) and so there is a \( F \in Y \) through which \( E \) was reached.
  - Because \( E \) is \( 0 \)-valent, \( F \) is \( 1 \)-valent:
    - One configuration is reached from the other, and by assumption \( F \) is not bivalent.
  - \( Y \) contains both \( 0 \)-valent and \( 1 \)-valent configurations.
Proof of lemma 3

Now let \( C, C_T \in \mathcal{X} \) such that \( D_T = a_K(C_T) \) is i-valent while \( C_5 = a_W(C_5) \) (w.l.o.g.) for some action \( a \) (i.e. they are 'neighbours')

- These \( D_T \) exist because \( \mathcal{Y} \) contains both 0 and 1 valent configurations

If \( p = q \) then \( D_T = a_W(D_T) \) by lemma 1, but this is impossible as this conflicts with valence of both configurations

If \( p = q \) let \( \sigma \) be any finite run in which \( q \) takes no steps and that leads to a decision when started in \( C_p \). Let \( \Lambda = \sigma(C_p) \).

By lemma 1, \( \sigma \) is also applicable to \( D \). Let \( E_T = \sigma(D) \). \( E_T \) is i-valent because \( D \) is.

By lemma 1, \( E_0 = a_K(\Lambda) \) and \( E_5 = a_W(a_W(\Lambda)) \). But then \( \Lambda \) is bivalent and not deciding, contradicting the choice of \( \sigma \).

Proof of impossibility result

Let \( C_0 \) be a bivalent initial configuration

- This one exists by lemma 2

Let \( C_T \) be the bivalent configuration at the start of stage \( i \)

- Let \( p \) be the processor that has taken a step longest ago that has an action \( a \) enabled:
  - Once enabled it will be enabled forever; hence the run constructed this way is fair and admissible

According to lemma 1 there is a bivalent configuration \( C_{i+1} \) reachable from \( C_i \) in which \( a \) takes the last step

Repeat forever, always staying bivalent