Tasks IIS

Solving a task in the iterated immediate snapshot model

Topological representation
Topological representation: one shot Immediate snapshot

p₁ sees {p₁, p₂}
p₂ sees {p₁, p₂}
p₃ sees {p₁, p₂, p₃}
p₃ sees {p₂, p₃}
p₂ sees {p₁, p₂, p₃}

p₁ sees {p₁}
p₂ sees {p₁, p₂, p₃}
p₃ sees {p₁, p₂, p₃}
p₂ sees {p₂, p₃}
p₃ sees {p₂, p₃}
Iterated Immediate Snapshot (IIS)

Shared variables:
\[ IS_1, IS_2, IS_3, \ldots \] // a series of one-shot IS

Each process \( p_i \) with input \( v_i \):
\[
\begin{align*}
  & r := 0 \\
  \text{while true do} \\
  & \hspace{1em} r := r+1 \\
  & \hspace{1em} v_i := IS_r.\text{WriteRead}_i(v_i)
\end{align*}
\]

\textbf{IS:} WriteRead\(_i(v_i)\)

Vectors \( S_1, \ldots, S_N \) satisfy:

- \textit{Self-inclusion:} for all \( i \): \( v_i \) is in \( S_i \)
- \textit{Containment:} for all \( i \) and \( j \): \( S_i \) is subset of \( S_j \) or \( S_j \) is subset of \( S_i \)
- \textit{Immediacy:} for all \( i \) and \( j \): if \( v_i \) is in \( S_j \), then \( S_i \) is a subset of \( S_j \)
From IIS to AS

We simulate an execution of full-information protocol (FIP) in the AS model, i.e., each process $p_i$ runs:

\[
\text{state} := \text{input value of } p_i \\
\text{repeat} \\
\quad \text{update}_i(\text{state}) \\
\quad \text{state} := \text{snapshot()} \\
\text{until} \ \text{decided}(\text{state})
\]

(the input value and the decision procedure depend on the problem being solved)

If a problem is solvable in AS, it is solvable with FIP

For simplicity, assume that the $k$-th written value $= k$ (“without loss of generality” – every written value is unique)
Iterated standard chromatic subdivision (ISDS)
ISDS: one round of IIS
two rounds of IIS
to solve task: IIS is equivalent to read-write
Consensus generalized: k-Set Agreement

A process \textit{proposes} an \textit{input} value in \( V \) (\(|V| \geq k+1\)) and tries to \textit{decide} on an \textit{output} value in \( V \)

\begin{itemize}
  \item \textit{k-Agreement}: At most \( k \) distinct values are decided
  \item \textit{Validity}: Every decided value is a proposed value
  \item \textit{Termination}: No process takes infinitely many steps without deciding
    \hspace{1cm} (Every \textit{correct} process decides)
\end{itemize}

1-set agreement = consensus
(N-1)-set agreement = \textit{set agreement}
Impossibility of Set Agreement

**Theorem 2** No *wait-free* algorithm solves set agreement in read-write model

- Starting with $N$ values, there is no way to drop one (decide on at most $N-1$)

- Implies the impossibility of wait-free $k$-set agreement for all $k \leq N-1$
Solving Set Agreement in IIS

- Each \( p_i \) proposes its id \( i \)
- There is an IIS round \( r \), such that each process (that reaches round \( r \)) outputs some id
- In each run, the set of output ids is a subset of participants of size \( \leq N-1 \)

Implies Sperner coloring of the subdivided simplex IIS\( r \)
Simplexes

- 0-simplex
- 1-simplex
- 2-simplex
- 3-simplex
Simplexes

Combinatorial: a set of vertexes.

0-simplex

1-simplex

2-simplex

3-simplex
Simplexes

Combinatorial: a set of vertexes.

Geometric: convex hull of points in general position

0-simplex

1-simplex

2-simplex

3-simplex
Simplicial Complex
Simplicial Complex

Combinatorial: a set of simplexxes close under inclusion.
Simplicial Complex

Combinatorial: a set of simplexes close under inclusion.

Geometric: simplexes “glued together” along faces …
For each simplex $s$ of $B$ there is a simplex $t$ of $A$ such that $|s| \subseteq |t|$. 

For each simplex $t$ of $A$, $|t|$ is the union of a finite set of geometric simplexes of $B$. 

**Subdivision** 

*B is a subdivision of A:*
“Chromatic” Subdivision

Simplex A

B Subdivision de A

From IIS
Sperner Coloring

“Corners” have distinct colors

Edge vertexes have corner colors

Every vertex has face boundary colors

Sperner Lemma: In any Sperner coloring, at least one $n$-simplex has all $n+1$ colors

Assume there is a wait-free protocol solving set agreement after $r$ rounds IIS we get a subdivision of $S^{n-1}$ each node is coloured by its decision, we get a Sperner coloring $\Rightarrow$ $N$ different values are decided
Sperner’s coloring of a subdivided simplex

IIS:
Each vertex of the original simplex $S^{N-1}$ (initial state) is colored with a process id

Sperner’s coloring:
Each vertex $v$ in the subdivision $D$ is colored with a distinct color:
- If the vertex belongs to a face $S$ of $S^{N-1}$, then it is in $\text{colors}(S)$
Sperner’s Lemma

Sperner’s coloring of any subdivision D of $S^{N-1}$ has a simplex with all N colors

IIS and decision on set agreement agreement defines a Sperner’s coloring. Then at one execution with N decided values!

Corollary: wait-free set agreement is impossible in IIS (and, thus, in read-write)
In at least one IIS run, N values are decided.

Here: suppose processes decide in one round.
Sperner’s lemma: inductive step

**Claim**: for each \( k=0,\ldots,N-1 \), face \( p_0,\ldots,p_k \) of \( S^{N-1} \) has an odd number of \( k \)-dimensional simplexes colored \( 0,\ldots,k \) (odd then at least one!)

By induction: \( k=0 \) - trivial (exactly one)

\( k=1 \), simple counting

Suppose the claim holds for \( k<N-1 \) and consider the face \( 0,\ldots,k \)
Sperner’s: rooms and doors

Each k-simplex contained in face \( p_0, \ldots, p_k \) is a **room**

A (k-1)-dimensional face (a subset of k-1 vertices) of a room colored in \( 0, \ldots, k-1 \) is a **door**

A door is an **exit** if it is contained in the boundary of \( p_0, \ldots, p_k \)
Sperner’s: rooms and doors

There is an **odd** number of exits! (Induction)

By Sperner’s coloring no faces other than in $p_0,\ldots,p_{k-1}$ can contain simplexes colored $0,\ldots,k-1$

✩ Exits may only be contained in $p_0,\ldots,p_{k-1}$

✩ By induction, $p_0,\ldots,p_{k-1}$ contains an odd number of doors (colored with $0,\ldots,k-1$)
Sperner’s: corridors and dead ends

Consider a room with a door (k vertices colored 0,…,k-1)

Two cases possible (depending on the color of the remaining vertex):

✶ the remaining vertex is colored k - the room has exactly one door (dead end)
  • the room is fully colored (0,…,k)

✶ The remaining vertex is colored in one of {0,…,k-1} - the room has two doors

We show that there is an odd number of dead ends contained in face $p_0,…,p_k$
Sperner’s: counting fully colored rooms

Start with an exit and walk through the doors
There are only finitely many rooms, thus, two only cases possible:

★ Stop in a dead end (fully colored simplex)
★ Reach another exit (two exits for such a walk)
Sperner’s: crossing the rooms

The number of exits is odd

=> The number of dead ends (fully colored simplexes) reachable from exit is \textbf{odd}!

Now start in a dead end \textbf{not reachable} from an exit: can only stop in another dead end \textbf{not reachable} from an exit

⇒ The number of dead ends \textbf{not reachable} from exit is \textbf{even}!

⇒ The total number of fully colored rooms is \textbf{odd}
Thus

- No algorithm can wait-free solve (N-1)-set agreement in IIS
  - otherwise Sperner’s coloring of some IIS\(r\) would have no fully colored simplexes (but there is at least one!)

- No algorithm can wait-free solve set agreement in AS (or any read-write model):
  - otherwise we can (non-blocking) simulate it in IIS and thus find a wait-free algorithm in IIS

- We cannot tolerate N-1 failures: can we tolerate less?
  - E.g., can we solve consensus (1-set agreement) \(1\)-resiliently?
So…

* No algorithm can wait-free (N-resiliently) solve consensus (proved!

* We cannot tolerate N-1 failures: can we tolerate less?
  • can we solve consensus 1-resiliently?
1-resilient consensus?

What if we have 1000000 processes and one of them can crash?

NO

We present a direct proof now
(an indirect proof by reduction to the wait-free impossibility also exists)
Theorem 2  No 1-resilient (assuming that one process might fail) algorithm solves consensus in read-write

Proof
By contradiction, suppose that an algorithm A solves 1-resilient binary consensus among $p_0, \ldots, p_{N-1}$
A configuration (or state) = state of each process + state of shared memory
Initial configuration = one input value per process + initial state of processes and shared memory
A step = atomic step (read or write on shared memory) of a process
A run of A is a sequence of steps applied to the initial state
A schedule is a (infinite) sequence of process ids $i_1, i_2, \ldots, i_k, \ldots$

A run of A can be seen as and initial input configuration (one input per process) and a schedule (or an initial sequence of a schedule)

Every correct (taking sufficiently many steps) process decides!
(t-resiliency means that in every run at most t processes take a finite number of steps)
Let $R$ be a finite run

- We say that $R$ is \textit{v-valent} (for $v$ in $\{0,1\}$) if $v$ is decided in \textit{every} infinite extension of $R$

- We say that $R$ is \textit{bivalent} if $R$ is neither 0-valent nor 1-valent (there exists a 0-valent extension of $R$ and a 1-valent extension of $R$)

- We say $R$ is \textit{univalent} if $R$ is not bivalent
Proof: valence claims

Claim 1 Every finite run is 0-valent, or 1-valent, or bivalent. (by Termination)

Claim 2 Any run in which some process decides v is v-valent (by Agreement)

Corollary 1: No process can decide in a bivalent run (by Agreement).
Bivalent input

Claim 3 There exists a bivalent input configuration (empty run)

Proof
Suppose not:
Consider sequence of input configurations $C_0, \ldots, C_N$:
in $C_i$: $p_0, \ldots, p_{i-1}$ propose 1, and $p_i, \ldots, p_N$ propose 0

- All $C_i$'s are univalent (hypothesis)
- $C_0$ is 0-valent (by Validity)
- $C_N$ is 1-valent (by Validity)
Bivalent input

There exists \( i \) in \( \{0, \ldots, N-1\} \) such that \( C_i \) is 0-valent and \( C_{i+1} \) is 1-valent! \( C_i \) and \( C_{i+1} \) differ only in the input value of \( p_i \) (it proposes 0 in \( C_i \) and 1 in \( C_{i+1} \)).

Consider a run \( R \) starting from \( C_i \) in which \( p_i \) takes no steps (crashes initially): eventually all other processes decide 0.

Consider \( R' \) exactly like \( R \) except that it starts from \( C_{i+1} \), \( R \) and \( R' \) are indistinguishable (\( p_i \) takes no steps).
- Thus, every process decides 0 in \( R' \) --- contradiction (\( C_{i+1} \) is 1-valent).
Claim 4 There exists a finite run \( R \) and two processes \( p_i \) and \( p_j \) such that \( R.i \) is 0-valent and \( R.j.i \) is 1-valent (or vice versa) (\( R \) is called critical)

Proof of Claim 4: By construction, First take the bivalent empty run \( C \) (by Claim 3 it exists)

Then Construct an ever-extending fair (giving each process enough steps) critical run
Proof of Claim 4: critical run

repeat forever
    take the next process \( p_i \) (in round-robin fashion)
    if for some \( R' \), an extension of \( R \), \( R'.i \) is bivalent then \( R := R'.i \)
    else stop

\* If never stops – ever extending (infinite) bivalent runs in which every process is correct (takes infinitely many steps) – contradiction with termination

\* If stops – (suppose \( R.i \) is 0-valent) – \( R \) is bivalent then consider a 1-valent \( R' \) extension of \( R \)
    - There is a critical configuration between \( R \) and \( R' \)
Proof (contd.): the next steps in $R$

(Let $p_0=i$ and $p_1=j$)

Four cases, depending on the next steps of $p_0$ and $p_1$ in $R$

- $p_0$ and $p_1$ are about to access different objects (registers) in $R$
- $p_1$ reads $X$ and $p_0$ reads $X$
- $p_0$ writes in $X$
- $p_1$ reads $X$
Proof (contd.): cases and contradiction

• \( p_0 \) and \( p_1 \) are about to access different objects in \( R \)
  - \( R.0.1 \) and \( R.1.0 \) are indistinguishable
Proof (contd.): cases and contradiction

* $p_0$ and $p_1$ are about to read the same object $X$

R.0.1 and R.1.0 are indistinguishable
Proof (contd.): cases and contradiction

* \( p_0 \) is about to write to \( X \)
  - Extensions of \( R.0 \) and \( R.1.0 \) are indistinguishable for all except \( p_1 \) (assuming \( p_1 \) takes no more steps)
Proof (contd.): cases and contradiction

* \( p_0 \) is about to read to \( X \)
  
  - Extensions of \( R.0.1 \) and \( R.1.0 \) are indistinguishable for all but \( p_0 \) (assuming \( p_0 \) takes no more steps)

\[
\begin{align*}
R & \\
p_0 \text{ reads } X & \quad p_1 \rightarrow X \\
p_1 \rightarrow X & \quad p_0 \text{ reads } X
\end{align*}
\]
Thus

⋆ No critical run exists
⋆ A contradiction with Claim 4

→ 1-resilient consensus is impossible in read-write
and message passing?

message passing:

• send/receive messages with asynchronous communication (each message sent by a process is eventually received by a correct process)
• process may crash (stop its execution)
• process p is correct if it doesn’t crash (makes an infinity of steps)
• t- resiliency at most t processes may crash
Message Passing versus Shared memory

simulating shared memory: with a majority of correct processes atomic registers may be implemented in asynchronous message passing models.

(Message-Passing and shared memory models are « equivalent » -with a majority of correct processes)
Simulating shared registers in message passing

simulation of a single-writer single-reader register

• assume a majority of correct processes
For the writer
to write$(v)$
\[ seq := seq + 1 \]
send $(W, v, seq)$ to all
wait until receiving $\lceil n/2 \rceil + 1$ messages $(ACK, seq)$

For the reader
to read()
send$(R)$ to all
wait until receiving $\lceil n/2 \rceil + 1$ messages $(V, v, s)$ such that $s > seq$
return \( val \)
such that $(V, val, S)$ has been received
and $S$ is the max of the sequence number of received V messages

For all processes
when $(W, v, s)$ is received
if $s > seq$ then
\[ val := v; seq := s \]
send $(ACK, s)$

when$(R)$ is received
send $(V, val, seq)$ to $p_r$
A majority of correct processes is needed

partition argument:

• if \( n \leq 2t \) then we can partition the set of processes in two set \( S_1 \) and \( S_2 \) such that \( |S_1| \geq t \) and \( |S_2| \geq t \).
  • Run \( A_1 \): all processes in \( S_1 \) are correct and all processes in \( S_2 \) are initially dead, \( p_0 \) invokes a write(1), at some time \( t_0 \) the write terminates
  • Run \( A_2 \): all processes in \( S_2 \) are correct and all processes in \( S_1 \) are initially dead, \( p_1 \) \( S_2 \) invokes a read(), at time \( t_0 + 1 \) the read terminates at time \( t_1 \)
  • Run \( B \): « merge » of \( A_1 \) and \( A_2 \) but no process crash. write(1) terminates before the read() and the read return 0

contradiction
Consensus in message passing

Same definition as in shared memory:

- Decision algorithm with:
  - Agreement (no two correct processes decide on different values)
  - Termination (all correct processes terminate)
  - Validity (the decision is an initial value).

There is no 1-resilient consensus algorithm in message passing.

- Partition argument if (t-resiliency) $t \leq n/2$ (Exercice)
- Shared memory enables to simulate message passing
primitives Rbcast Rdeliver

- **Agreement**: if correct process p Rdeliver m then every correct process Rdeliver m
- **Validity**: If correct process p Rbcast m then p Rdeliver m
- **Integrity**: If p Rdeliver m then there is a process q that has Rbcast m
RBcast

Algorithm for process $p$:

To execute $\text{Rbcast}(m)$
  send $(m)$ to $p$

$\text{Rdeliver}(m)$ occurs when
  \textbf{upon} receive$(m)$ \textbf{do}
    \textbf{if} has not previously executed $\text{Rdeliver}(m)$
    \textbf{then}
      send $(m)$ to all
      $\text{Rdeliver}(m)$
primitives \( \text{ABcast} \ \text{ABdeliver} \):

- \( \text{RBcast} \) properties:
- \textit{Total order:} If \( p \) and \( q \) \( \text{ABdeliver} \) \( m \) and \( m' \) then if \( p \) \( \text{ABdelivers} \) \( m \) before \( m' \) then \( q \) \( \text{ABdelivers} \) \( m \) before \( m' \)

\( \text{ABCast} \) is « universal »: (very informal) (active replication)

- state machine replication:
  - any sequential state machine \( A \)
  - \( t+1 \) processes simulate \( A \)
  - each request is made by atomic broadcast
  - then we get a \( t \)-resilient implementation of \( A \)
Réplication active et passive

### Active Replication (SM)

- **Client** sends op to all replicas
- Requests get ordered
- Execution
- All reply to client

### Passive Replication

- **Client** sends op to primary
- Only primary executes
- Primary forwards changes
- Primary replies to client
réplication passive

« Primary-Backup » (très simplifié)

• Un seul serveur est actif (primary ou maître)
• Les autres sont passifs (backup ou esclaves) et enregistrent passivement l'état du « primary »
• Si un primary tombe en panne...
  • Choisir un nouveau « primary » parmi les backups
  • Démarrer le primary à partir de l'état de backup
• Problème: Assurer l'atomicité des requêtes (Que se passe-t-il si le primary tombe en panne pendant une requête?)
réplication Active

Tous les server sont actifs et traitent toutes les requêtes dans le même ordre

- Tous les serveurs passent par les mêmes états (ils ont la même histoire)

Diffusion atomique:

- Chaque requête arrive dans le même ordre sur chaque processus (serveur): (1) tous les processus serveurs ont la même histoire (2) les réponses aux requêtes sont les mêmes
Algorithm for process $p$:

*Initialization:*

\[ R_{\text{Delivered}} := \emptyset \]
\[ A_{\text{Delivered}} := \emptyset \]

To execute `Abcast(m)`
\[ R_{\text{broadcast}}(m) \]

Adeliver(·) occurs when
\[
\text{upon } R_{\text{deliver}}(m) \text{ do }
\]
\[ R_{\text{Delivered}} := R_{\text{Delivered}} \cup \{m\} \]

**do forever**

\[ A_{\text{Undelivered}} := R_{\text{Delivered}} - A_{\text{Delivered}} \]
\[ \text{if } A_{\text{Undelivered}} \neq \emptyset \text{ then} \]
\[ k := k + 1 \]
\[ \text{propose}(k, A_{\text{Undelivered}}) \]
\[ \text{wait for } \text{decide}(k, \text{msgSet}) \]
\[ \text{batch}(k) := \text{msgSet} - A_{\text{Delivered}} \]

A-deliver all messages in \[ \text{batch}(k) \] in some deterministic order
\[ A_{\text{Delivered}} := A_{\text{Delivered}} \cup \text{batch}(k) \]
Then

Consensus (and reliable broadcast) enable Atomic broadcast

Atomic Broadcast enable consensus  Consensus? (easy -exercice)

Atomic broadcast and consensus are equivalent in message passing with crash failure

Then Consensus is « universal » in message passing with crash failure