Consensus with Partial Synchrony

Pedro Ferreira do Souto

Departamento de Engenharia Informática
Faculdade de Engenharia
Universidade do Porto

Outline

Outline

1. Failure Detection
2. Consensus
   - Problem Definition
   - Solution by Transformation of Synchronous Algorithms
   - PSynchAgreement
   - More Partially Synchronous Models
3. Further Reading

Outline

1. Failure Detection
2. Consensus
   - Problem Definition
   - Solution by Transformation of Synchronous Algorithms
   - PSynchAgreement
   - More Partially Synchronous Models
3. Further Reading

Outline

1. Failure Detection
2. Consensus
   - Problem Definition
   - Solution by Transformation of Synchronous Algorithms
   - PSynchAgreement
   - More Partially Synchronous Models
3. Further Reading

PSynchFD Failure Detector

Failure detector TIOA with bounds

stop$_i$ input actions

inform-stopped$(j)_i, i \neq j$ output actions, which notifiy process i that process j has stopped.

PSynchFD failure detector algorithm

1. Each process P_i continually sends messages to all the other processes.
2. If a process P_i performs a sufficiently large number m of steps without receiving a message from P_j, it records that P_j has stopped and outputs *inform-stopped*$(j)_i$
   - The number m of steps is taken to be the smallest integer that is strictly greater than $(d + \ell_2)/\ell_1 + 1$

Perfect failure detector reports

1. only failures that have actually happened;
2. all such failures to all other non-faulty processes.

Theorem 25.1: PSynchFD is a perfect failure detector

Proof (by contradiction)

It should be clear that all failures are eventually detected.

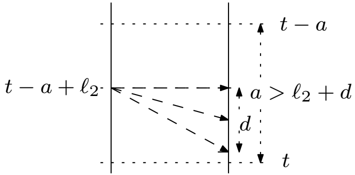
So, let's assume that P_i reports that P_j has stopped but it has not.

1. If P_i outputs *inform-stopped*$(j)_i$, it must have been the case that it has not received a message from P_j in the previous $(d + \ell_2)/\ell_1 + 1$ steps.
2. Since each step takes at least ℓ_1 time units, this means that strictly more than $d + \ell_2$ time units have passed since the last time P_i received a message from P_j.
3. Since the channel delay is at most d, then P_j has not sent a message for at least ℓ_2 time units.
4. Since P_j sends messages to every processes once per step, P_j has taken more than ℓ_2 to execute a step.
5. This is a contradiction, because ℓ_2 is the upper bound for P_j to take a step. Thus P_j must have stopped.

Lower bound on PSynchFD (Theorem 25.2 part 1)

Theorem 25.2 part 1

In any execution, the time from a *stop*$_j$ event until a *inform-stopped*$(j)_i$ event, if any, is strictly greater than d



Let t be the time when event *inform-stopped*$(j)_i$ occurs.

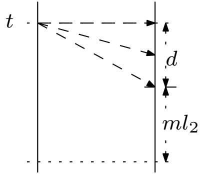
1. As pointed out above, it must be the case that P_i has not received any message from P_j for time $a > \ell_2 + d$.
2. Hence, it must be the case that P_j has not sent any message from $[t-a, t-a+\ell_2]$, for otherwise it would have been received by P_i in the interval $[t-a, t-a+\ell_2+d]$, which is included in $[t-a, t]$
3. Thus it must be the case that P_j has stopped by $t-a+\ell_2$. Since $a > \ell_2 + d$, it follows that $t - a + \ell_2 < t - d$, i.e. at least d time units before *inform-stopped*$(j)_i$.

Note This means that if P_i times out P_j, then all the messages P_j has sent before failing must have already been received.

Upper bound on PSynchFD (Theorem 25.2 part 2)

Theorem 25.2 part 2

In any admissible execution in which *stop*$_j$ event occurs, within time $Ld + d + O(L\ell_2)$ after *stop*$_j$, either an *inform-stopped*$(j)_i$ event or a *stop*$_i$ event occurs.



- $L = \ell_2/\ell_1$ is a measure of the uncertainty of process execution speeds.

Let t be the time when event *stop*$_j$ occurs.

1. Then no message is sent from P_j to P_i after time t, so no message is received by P_i from P_j after time $t + d$.
2. After receiving P_j's last message, P_i counts m steps, each of which can take at most ℓ_2 time to execute.
3. Because m is strictly greater than $(d + \ell_2)/\ell_1 + 1$, we get $m\ell_2 > (d + \ell_2)L + \ell_2$, i.e. $m\ell_2 = Ld + O(L\ell_2)$.
4. Thus, if P_i does not fail in the meantime, the total time from *stop*$_j$ to *inform-stopped*$(j)_i$ is $Ld + d + O(L\ell_2)$

Outline

1. Failure Detection

2. Consensus
   - Problem Definition
   - Solution by Transformation of Synchronous Algorithms
   - PSynchAgreement
   - More Partially Synchronous Models

3. Further Reading

Outline

Consensus: External interfaces

System A

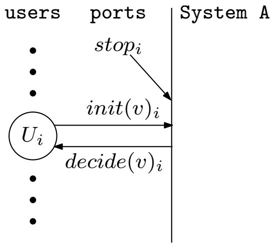
$init(v)_i$ input action;

$decide(v)_i$ output action;

$stop_i$ input action;

where $1 \leq i \leq n$ and $v \in V$

Note all actions with subscript i are said to occur on port i;



User U_i

$decide(v)_i$ input action;

$init(v)_i$ output action;

U_i performs at most one $init_i$ action in any timed execution.

Definition A sequence of $init_i$ and $decide_i$ actions is **well-formed** for i provided that it is some prefix of a sequence of the form $init(v)_i, decide(w)_i$.

Consensus: Problem definition (1/2)

Well-formedness: In any timed execution of the combined system, and for any port i, the interactions between U_i and A are well-formed for i.

Agreement: In any timed execution, all decision values are identical.

Validity: In any timed execution, if all *init* actions that occur contain the same value v, then v is the only possible decision value.

Failure-free termination: In any admissible failure-free timed execution in which *init* events occur on all ports, a *decide* event occurs on each port.

f-failure termination, $0 \leq f \leq n$: In any admissible timed execution in which *init* events occur on all ports, if there are *stop* events on at most f ports, then a *decide* event occurs on all the remaining ports.

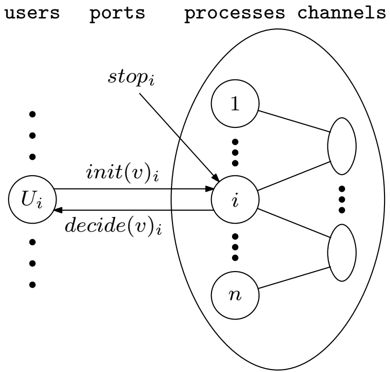
Definition **Wait-free termination** is the special case of f-failure termination where $f = n$.

Consensus: Problem definition (2/2)

System A Is the composition of the following TIOA with bounds

P_i with bounds ℓ_1 and ℓ_2 for each of its tasks, where $0 < \ell_1 \leq \ell_2 < \infty$. **Processes are subject to stopping failures.**

C_{ij} which are point-to-point reliable FIFO channels with an upper bound of d on the delivery time **for every message**



Definition A **solves the agreement problem** if it satisfies well-formedness, agreement, validity and failure-free termination.

Outline

1. Failure Detection
2. Consensus
   - Problem Definition
   - Solution by Transformation of Synchronous Algorithms
   - PSynchAgreement
   - More Partially Synchronous Models
3. Further Reading

Idea for a Solution

Main result

It is possible to solve agreement with f failures in the partially synchronous setting with upper and lower bounds of $f+1$ rounds (just like in the synchronous model).

Observation

All the algorithms for agreement in the synchronous network model require $f+1$ rounds to tolerate f stopping failures.

Idea

Transform these algorithms to algorithms in the partially synchronous network model.

Transformation of synchronous network algorithms (1/3)

Let A be any synchronous network algorithm for a complete graph network. The algorithm A' for the partially synchronous network model is as follows:

- Each process P_i is the composition of two TIOA with bounds:

  Q_i is i's portion of the PSynchFD algorithm. It includes:

  $stop_i$ input action.

  $informed\text{-}stopped_i$ output actions.

  R_i is the main automaton. It includes:

  $informed\text{-}stopped_i$ inputs (which are matched with Q_i outputs);

  stopped state variable, that keeps track of the set of failed processes, i.e. processes j for which it has received the inputs $informed\text{-}stopped(j)_i$;

  simulated state variables of process i of A.

Transformation of synchronous network algorithms (2/3)

Round r simulation

TIOA R_i executes the following steps:

1. i. Determines all its round r messages using the $msgs_i$ function from A and the current A's simulated state
   ii. Sends out these messages to their destination, using one task per destination process.
2. waits until it has received either
   - a round r message from R_j, or
   - an $\textit{inform-stopped}(j)_i$ input from Q_i

   from each $j \neq i$
3. Determines the new simulated state by applying $trans_i$ from A to the current simulated state and the messages received in round r (using *null* for the messages of processes in *stopped*).

Transformation of synchronous network algorithms (3/3)

Input/Output adaptation

- In A inputs appear in the initial states, and outputs are written to write-once local variables.
- So, we need to modify A' to obtain algorithm B:
    1. R_i does not begin the simulation of A until it receives an $init(v)_i$ input, at which time it initializes A's simulated state.
        - ⋆ But Q_i begins its timeout activity at the start of the timed execution.
    2. When R_i simulates the write of value v to its simulated output variable, it immediately after performs a $decide(v)_i$ output action.

Upper bound: Theorem 25.3

Theorem 25.3

1. B solves the agreement problem in the partially synchronous network model, and guarantes f-*failure* termination.
2. In any admissible timed execution in which inputs arrive at all ports and at most f failures occur, the time from the last *init* event until all nonfaulty process have decided is at most $f(Ld + d) + d + O(fL\ell_2)$

Proof (part 1) It is easy to see that B simulates A, and therefore solves the agreement problem.

Upper bound: Proof Theor. 25.3 part 2 (1/2)

Let α be an admissible timed execution of B.
Let $S = Ld + d + O(L\ell_2)$ be an upper bound for the *PSynchFD* algorithm.
Let $T(0), T(1), T(2), \ldots$ be a sequence of times, where $T(r)$ is defined as follows:

$T(0)$ is the time at which the last *init* occurs in α

$$T(1) = \begin{cases} T(0) + \ell_2 + S, \text{if some process fails by } T(0) + \ell_2 \\ T(0) + \ell_2 + d, \text{otherwise} \end{cases}$$

And for $r \geq 2$:

$$T(r) = \begin{cases} T(r-1) + \ell_2 + S, & \text{if some process fails in the time interval} \\ & (T(r-2) + \ell_2, T(r-1) + \ell_2] \\ T(r-1) + \ell_2 + d, & \text{otherwise} \end{cases}$$

Claim 25.5

For all $r \geq 0$, $T(r)$ is an upper bound on the time for all not-yet-failed processes to complete their simulation of r rounds of A.

Upper bound: Proof Theor. 25.3 part 2 (2/2)

- $T(f+1)$ is an upper bound for all not-yet-failed processes to complete their simulation of $f+1$ rounds.
- $T(f+1)+O(\ell_2)$ is an upper bound on the time for all nonfaulty processes to perform their *decide* output action.
- From the definition of $T(r)$:

$$T(f+1) = T(0) + (T(1) - T(0)) + \ldots + (T(f+1) - T(f))$$

- Given that there are at most f faults, and $S > d$ we have:

$$T(f+1) \leq T(0) + f(\ell_2 + S) + (\ell_2 + d)$$

- Plugging in the bound for S $(= Ld + d + O(L\ell_2))$ yields:

$$T(f+1) \leq T(0) + f(Ld + d) + d + O(fL\ell_2)$$

Which implies the upper bound.

Upper bound: Proof of Claim 25.5

Claim 25.4

Let $r \geq 0$ and let j be any process index.
If process j fails by time $T(r) + \ell_2$, then j is detected as failed by all not-yet-failed processes by time $T(r+1)$

Proof S is an upper bound for the time to detect process failures.

Proof of Claim 25.5 (by induction on r)

Basis $r = 0$: trivially true.

Inductive step $r \geq 1$.

1. If some process j fails by time $T(r-1) + \ell_2$, then Claim 25.4 implies that it is timed out by all not-yet-failed processes by $T(r)$.
2. Otherwise, it sends all its round r messages by $T(r-1) + \ell_2$. These arrive at their destinations by time $T(r-1) + \ell_2 + d$.

Lower bound: Theorem 25.6

Theorem 25.6

Suppose that $n \geq f + 2$. Then, there is no n-process agreement algorithm for the partially synchronous network model that guarantees f-failure termination, in which **all** non-faulty processes **always** decide strictly before time $(f + 1)d$.

Idea of proof - by contradiction

This theorem extends the lower bound of $f + 1$ on the number of rounds to solve agreement in the synchronous network model to the partially synchronous network model.

1. Assume there is such an algorithm A.
2. Transform A into an f-round synchronous algorithm A', thus contradicting a previously proved result.

Lower bound: proof sketch of Theor. 25.6 (1/4)

Since this is an impossibility result, we will consider a **strongly timed model**, i.e. a partially synchronous model whose executions have the following restrictions:

1. All inputs arrive at the beginning, i.e. time 0.
2. All tasks proceed as slowly as possible, subject to the ℓ_2 upper bound.
    - All locally controlled steps occur at times that are multiples of ℓ_2.

   For each process, the task steps occur in a prespecified order.
3. For every $r \in \mathbb{N}$, all messages sent in the interval $[rd, (r+1)d)$ are delivered at exactly time $(r+1)d$.
   Also, messages delivered to a single process i at the same time, are delivered in order of the sender indices.
4. At a time that is multiple of both ℓ_2 and d, all the message deliveries occur prior to all the locally controlled process steps.

Lower bound: proof sketch of Theor. 25.6 (2/4)

- WLOG, let A be a "deterministic" algorithm that solves agreement in the strongly timed model.
- Since messages are delivered at times multiple of d and processes must decide before $(f+1)d$, let processes decide at their first step after the time fd message deliveries (we assume $\ell_2 < d$)
- The behavior of A is very close to the behavior of an f-round synchronous network algorithm:
  - For every $r \geq 1$, since no message arrives between times $((r-1)d, rd)$, the messages sent in the interval $[(r-1)d, rd)$ are all determined by process states just after the time $(r-1)d$ deliveries. Thus we might try to regard these messages as the round r of a synchronous algorithm.

Lower bound: proof sketch of Theor. 25.6 (3/4)

Problem The assumptions wrt process failures are not identical.

strongly timed model if process i fails at some point in interval $[(r-1)d, d)$, then for each node $j \neq i$, it may succeed in sending some of the messages it is supposed to send and fail to send the remaining. In the

synchronous network model if process i fails during round r, then, for each process j, it either fails or succeeds to send round r message

- This is equivalent to assume that, for each process j, i sends either all or none of its messages in the interval $[(r-1)d, r)$.

Solution Generalize the synchronous network model in a way that does not invalidate the proof of its lower bound for reaching consensus in the synchronous network model (Theorem 6.33):

- We allow process i to send, at each round r, a finite sequence of messages, each to an arbitrary, specified destination.
- Instead of sending only one message to each process.

Lower bound: proof sketch of Theor. 25.6 (4/4)

- It is possible to transform algorithm A into an agreement algorithm A' in this stronger synchronous model:
  - The sequence of messages process i sends in the interval $[(r-1)d, rd)$ in A, is the sequence of messages A' sends in its round r.
  - The behavior caused by the failure of i in A corresponds to a possible behavior in A'.

  The resulting algorithm A' is an f-round agreement algorithm for the stronger synchronous model, for $n \geq f + 2$. This is a contradiction of Theorem 6.33:
  - Suppose that $n \geq f + 2$. Then there is no n-process stopping-agreement algorithm that tolerates f faults, in which nonfaulty processes always decide by the end of round f

Outline

PSynchAgreement: Rationale

- The bounds $(((f+1)d, fLd + (f+1)d))$ for the B algorithm are not very tight.
  - Furthermore, the upper bound is somewhat large.
- *PSynchAgreement* is a more efficient algorithm that uses the *PSynchFD* failure detector just like the B algorithm. I.e., each process P_i is composed of 2 TIOA with bounds:

  $\mathbf{Q_i}$ is i's portion of the PSynchFD algorithm

  $\mathbf{R_i}$ is the main automaton. It includes the following:

  informed-stopped$_i$ inputs (which are matched with Q_i outputs);

  stopped state variable, that keeps track of the set of failed processes, i.e. processes j for which it has received the inputs $\textit{informed-stopped}(j)_i$;

PSynchAgreement: Algorithm (1/2)

- *PSynchAgreement* proceeds in rounds, numbered $0, 1, \ldots$
  - In each round, R_i tries to reach a decision.
  - R_i can decide 0 only in even numbered rounds.
  - R_i can decide 1 only in odd numbered rounds.
- R_i begins round 0 only after it receives its input.
- R_i maintains a variable *decided* to keep track of the processes from which it has received a *decided* message.

Round 0

If R_i's input is 0, then R_i does the following:

1. send *goto*(2) message to all processes
2. output *decide*(0)
3. send *decided* to all processes

If R_i's input is 1, then R_i does the following:

1. send *goto*(1) message to all processes
2. go to round 1

PSynchAgreement: Algorithm (2/2)

Round r (> 0)

1. R_i waits until it has received, either a:

   goto$(r+1)$ message from some process, or

   goto(r) message from every process that is not in $stopped_i \cup decided_i$.
2. If R_i has received a *goto*$(r+1)$ message, then it does the following:
   1. send *goto*$(r+1)$ message to all processes
   2. go to round $r+1$

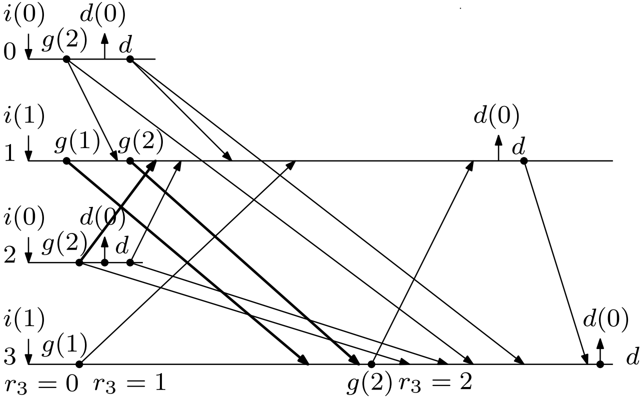
   Else (R_i has received only *goto*(r) messages),
   1. send *goto*$(r+2)$ message to all processes
   2. output *decide*$(r \mathit{mod}\, 2)$
   3. send *decided* to all processes

Note The algorithm is biased towards decision value 0.

Definition A process i **tries to decide** at round $r \geq 0$ if it sends at least one *goto*$(r+2)$ message in preparation for a *decide* event at round r.

- i may end not performing *decide* if it fails in the meantime

PSynchAgreement: "Cleaned up" execution



- The *goto*(2) message sent by P_2 in its round 0 is received by P_1 when it is already in round 2.
- In round 1, P_3 receives a *goto*(2) message, which was relayed by P_1, after having received a *goto*(1) message also from P_1.

PSynchAgreement: Proof of safety properties (1/2)

Lemma 25.7

In any timed execution of *PSynchAgreement* and for any $r \geq 0$, the following is true:

1. If any process sends a $goto(r+2)$ message, then some process tries to decide in round r.
2. If any process reaches round $r+2$, then some process tries to decide at round r.

Lemma 25.8

In any timed execution of *PSynchAgreement* and for any $r \geq 0$, if a process i decides at round r, then the following are true:

1 R_i sends no $goto(r+1)$ messages.

2 R_i sends a $goto(r+2)$ message to every process.

3 No process tries to decide at round $r+1$.

PSynchAgreement: Proof of safety properties (2/2)

Theorem 25.9: Safety properties

The *PSynchAgreement* algorithm guarantees **well-formedness, agreement** and **validity**.

Well-formedness is straightforward.

Validity

- If all processes start with 0, then no process ever leaves round 0, and because this is an even round cannot decide 1.
- If all processes start with 1, then no process tries to decide 0 in round 0. From Lemma 25.7 part 1, no process reaches round 2, or any other even round. Thus no process decides 0.

Agreement Suppose that R_i decides at round r and that no process decides at any earlier round.

By Lemma 25.8 part 3, no process tries to decide at round $r + 1$.

Then by Lemma 25.7 part 1, no process can reach round $r + 3$, and so on. Thus a process can reach only rounds with the same parity as r, hence all the decisions must be the same.

PSynchAgreement: Proof of Lemma 25.7

Lemma 25.7

In any timed execution of *PSynchAgreement* and for any $r \geq 0$, the following is true:

1. If any process sends a $goto(r+2)$ message, then some process tries to decide in round r.
2. If any process reaches round $r+2$, then some process tries to decide at round r.

Proof

1. The first $goto(r+2)$ message must be generated this way. Other $goto(r+2)$ messages are generated after receiving such a message.
2. A process advances to round $r+2$ only after receiving a $goto(r+2)$ message.

PSynchAgreement: Proof of Lemma 25.8

Proof (parts 1 and 2) Should be clear from the algorithm specification.

Proof (part 3) **By contradiction.** Assume R_j tries to decide at round $r+1$. Then at some point in round $r+1$:

- R_j must have received only $goto(r+1)$ messages and no $goto(r+2)$ messages from all processes that are not in $stopped_j \cup decided_j$.
- Since i sends no message $goto(r+1)$, then it must be in $stopped_j \cup decided_j$.
- If $i \in stopped_j$, then by the upper bound on *PSynchFD*, R_j must have already received all messages sent by R_i before it failed. But, then it should have also received a $goto(r+2)$ message, which is a contradiction.
- If $i \in decided_j$, then R_j must have received a *decided* message from R_i. But, R_i sends such a message only after sending a $goto(r+2)$ message. Because the channels are FIFO, then R_j must have already received the $goto(r+2)$ message, which is a contradiction.

PSynchAgreement: Lemma's for Liveness

Definition A round r is **quiet** if there is **some** process that does not receive a $goto(r+1)$ message from any other process.

Lemma 25.10

In any admissible execution of *PSynchAgreement*, each process continues to advance from round to round until it either fails or decides.

Lemma 25.13

In any admissible execution of *PSynchAgreement* in which there are at most f failures, there is a quiet round numbered at most $f+2$.

Lemma 25.12

In any admissible execution of *PSynchAgreement*, if round r is quiet, then no process ever advances to round $r+1$.

PSynchAgreement: Wait-free termination

Theorem 25.14

The *PSynchAgreement* algorithm guarantees wait-free termination, i.e. that all nonfaulty processes eventually decide, for any $0 \leq f \leq n$ faulty processes.

Proof Consider an admissible timed execution in which all *init* events occur. Let i be any nonfaulty process.

- By Lemma 25.10, R_i continues to advance from round to round until it decides.
- But Lemma 25.13 implies that there is some quiet round r.
- And Lemma 25.12 implies that R_i cannot advance to round $r + 1$.
- Therefore R_i must decide by round r.

Skip lemma proofs

PSynchAgreement: Proof Lemma 25.10

Lemma 25.10

In any admissible execution of *PSynchAgreement*, each process continues to advance from round to round until it either fails or decides.

Proof **By contradiction.** Let r be the first round at which process i gets stuck. Note that r must be at least 1.

- For any process P_j that ever fails, Q_i must eventually detect its failure and R_i will put j in $stopped_i$.
- Also, for any process P_j that ever decides but never fails, R_i must eventually receive its *decided* message and put j in $decided_i$.
- Let I be the set of the remaining processes.
- Because r is the first round at which some process gets stuck, then all processes in I must eventually reach round r or $r+1$.
- Since $r \geq 1$, then all processes in I must send either a $goto(r)$ or $goto(r+1)$ message to P_i, which R_i eventually receives.
- Thus the condition for R_i to either decide or move to round $r+1$ is satisfied, **i.e.** i **does not get stuck at round** r.

PSynchAgreement: Proof Lemma 25.13 (1/2)

Lemma 25.13

In any admissible execution of *PSynchAgreement* in which there are at most f failures, there is a quiet round numbered at most $f + 2$.

Lemma 25.11

In any admissible execution of *PSynchAgreement* and for $r \geq 0$, the following are true:

1. If no process tries to decide at round r, then round $r + 1$ is quiet.
2. If some process decides at round r, then round $r + 2$ is quiet.

Remember A round r is **quiet** if there is **some** process that does not receive a $goto(r + 1)$ message from any other process.

PSynchAgreement: Proof Lemma 25.13 (2/2)

Proof

1. If any process decides by round f, then this follows from Lemma 25.11, part 2.
2. Suppose that no process decides by round f.
   Since there are at most f failures, there must be some round r, $0 \leq r \leq f$, in which no process fails.
   We claim that no process tries to decide in round r.
   Thus, it follows from Lemma 25.11, part 1 that round $r + 1$ is quiet.

   ### Proof of claim

   - Suppose for the sake of **contradiction** that some process i tries to decide in round r.
   - Since process i does not fail at round r, admissibility implies that process i must decide at round r.
   - But this contradicts the assumption that no process decides by round f.

PSynchAgreement: Proof of Lemma 25.12

Lemma 25.12

In any admissible execution of *PSynchAgreement*, if round r is quiet, then no process ever advances to round $r+1$.

Remember A round r is **quiet** if there is **some** process that does not receive a $goto(r+1)$ message from any other process.

Proof (by contradiction)

If process i advances to round $r+1$, then R_i has previously sent a $goto(r+1)$ message to all processes. These eventually receive them, which means that round r is not quiet.

PSynchAgreement: Proof of Lemma 25.11

Lemma 25.11

In any admissible execution of *PSynchAgreement* and for $r \geq 0$, the following are true:

1. If no process tries to decide at round r, then round $r + 1$ is quiet.
2. If some process decides at round r, then round $r + 2$ is quiet.

Remember A round r is **quiet** if there is **some** process that does not receive a $goto(r + 1)$ message from any other process.

Proof

1. From Lemma 25.7, part 1, if no process tries to decide in round r, then no process sends a $goto(r + 2)$ message, and therefore round $r + 1$ is quiet.
2. From Lemma 25.8, part 3, if some process decides at round r, then no process tries to decide at round $r + 1$. Then, part 1 implies that round $r + 2$ is quiet.

PSynchAgreement: Upper bound

Theorem 25.15

In any admissible timed execution of *PSynchAgreement* in which inputs arrive on all ports and there are at most f failures, the time from the last *init* event until all nonfaulty processes have decided is at most $Ld + (2f + 2)d + O(f\ell_2 + L\ell_2)$

Proof The proofs of Theorem 25.14 and its supporting lemmas have shown that:

1. The execution must consist of:
   - A sequence of non-quiet rounds, numbered up to $f + 1$
   - Followed by a single quiet round, say r.
2. All nonfaulty processes must decide without advancing past round r.

PSynchAgreement: Proof of upper bound (1/2)

Let $S = Ld + d + O(L\ell_2)$ be an upper bound for the *PSynchFD* algorithm.
Let T', $T(0)$, $T(1)$, $T(2), \ldots, T(r)$ be a sequence of times, where, T' is the time at which the last *init* occurs

$T(k)$ with $0 \leq k \leq r$, is the latest time by which every process has either failed, decided, or advanced to the next round, $k+1$.

Thus, all nonfaulty processes must decide by $T(r)$

Clearly:

$T(0) - T' = O(\ell_2)$ is the time for round 0.

$T(k) - T(k-1) \leq S + O(\ell_2)$, with $k \geq 1$ is an upper bound for round k.

Plugging in the value for S we get:

$$T(k) - T(k-1) \leq Ld + d + O(L\ell_2)$$

We claim (Claim 25.16) that for non-quiet rounds not depend on L:

$$T(k) - T(k-1) \leq (f_k + 1)(d + O(f_k\ell_2))$$

where f_k denotes the number of processes that fail while sending $goto(k+1)$ messages.

PSynchAgreement: Proof of upper bound (2/2)

Since:

$$T(0) - T' = O(\ell_2)$$
$$T(k) - T(k-1) \leq (f_k + 1)(d + O(f_k \ell_2)), \text{ for all } k, 1 \leq k \leq r-1$$
$$T(r) - T(r-1) \leq Ld + d + O(L\ell_2)$$

It follows that:

$$T(r) - T' \leq Ld + d + O(L\ell_2) + \sum_{k=1}^{k=r} (f_k + 1)(d + O(\ell_2))$$

Finally, since $\sum_{k=1}^{k=r-1} f_k \leq f$ and $r \leq f + 2$, we obtain:

$$T(r) - T' \leq Ld + (2f + 2)d + O(f\ell_2 + L\ell_2)$$

PSynchAgreement: Proof of bound for non-quiet round

Claim 25.16

Let f_k denote the number of processes that fail while sending $goto(k+1)$ messages. Then the total time that elapses from the sending of the first $goto(k+1)$ message by R_j until the receipt of the $goto(k+1)$ message by R_i is at most $(f_k+1)d + O(f_k \ell_2)$

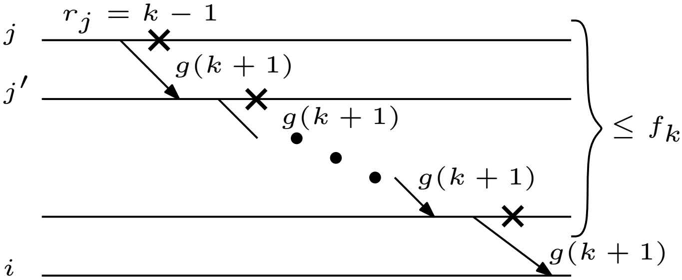
- Since R_j sends the first $goto(k)$ while in round $k-1$, it follows that it is sent before $T(k-1)$
- From Claim 25.16, it follows that all processes either advance to round $k+1$, fail, or decide by:

$$T(k-1)+(f_k+1)d+O(f_k\ell_2)+O(\ell_2) = T(k-1)+(f_k+1)(d+O(\ell_2)$$

- Thus, from the definition of $T(k)$, for any non-quiet round:

$$T(k) - T(k-1) \leq (f_k+1)(d+O(f_k\ell_2))$$

PSynchAgreement: Proof of Claim 25.16 (1/2)



Proof R_j sends its *goto*$(k+1)$ messages as part of an attempt to send such messages to all processes including P_i.

1. If P_j does not fail in the middle of this attempt, then R_j succeeds in sending this message to R_i, and R_i will receive it within time d of when R_j sends it.

PSynchAgreement: Proof of Claim 25.16 (2/2)

Proof

2. Even if P_j fails in the middle of this attempt, all the messages it succeeds in sending will arrive to their destination within time d of when R_j sends it.
   - Likewise, each process $P_{j'}$ that relays the message from R_j to R_i sends its $goto(k+1)$ message as part of an attempt to send such message to all processes including to P_i.
   - Again, if P'_j does not fail in the middle of its attempt, then R'_j succeeds in sending the message to R_i, which receives it within time d ...
3. Because the maximum number of faulty nodes in round k is f_k, the total time from when the original $goto(k+1)$ message is sent by R_j until i receives some $goto(k+1)$ message is at most $(f_k+1)d + O(f_k \ell_2)$.

More On bounds

Theorem 25.17

Suppose that $n \leq f+1$. Then there is no n-process agreement algorithm for the partially synchronous model that guarantees f-failure termination, in which all non-faulty processes always decide strictly before time $Ld+(f-1)d$.

Proof See Section 25.5 of Nancy Lynch's book.

	Lower bound	Upper bound
Transformed Synchronous Algorithm	$(f+1)d$	$fLd+(f+1)d$
PSynchAgreement		$Ld+2(f+1)d$
	$Ld+(f-1)d$	

Discussion

Question Is the $f+1$ bound on the number of rounds surprising?

Answer Shouldn't be!

- Although the model used considers the time explicitly, the system is still synchronous, not partially synchronous.
    1. We assumed an upper bound, d, on the time a channel takes to deliver a message.
    2. We assumed both a lower bound, ℓ_1, an an upper bound, ℓ_2, on the time a process takes to execute an action.
- These are the requirements often stated in the definition of a synchronous system
    - Usually, together with access to a clock, that measures the time within a linear envelope of "real" time.
    - What happened to this assumption?

Outline

Some results for *truly* partially synchronous systems

Synchronous processes, asynchronous channels I.e., the time taken by channels to deliver a message is unbounded.

Theorem 25.23

There is no algorithm in the model with synchronous processes and asynchronous channels that solves the agreement problem and guarantees 1-failure termination.

Asynchronous processes, synchronous channels I.e., the time taken by processes to take an action is unbounded.

Theorem 25.24

There is no algorithm in the model with asynchronous processes and d-bounded channels that solves the agreement problem and guarantees 1-failure termination.

Proof sketches By contradiction. The behavior observed may be the same as in a totally asynchronous system. Thus ...

Some results for *eventually* synchronous systems

Definition eventually both the processes and the channels take a bounded time to execute their actions.

- E.g., both processes and channels may "sleep" for an arbitrary finite time, after which they start behaving synchronously.

Result In this case, there is a solution.

- But it requires $n > 2f$. The intuition is as follows:
  - To ensure termination, a process should not wait for messages from more than $n - f$ responses, because up to f nodes may fail.
  - To ensure agreement, every decision should take into account the messages from at least one common process.

Theorem 25.25

The agreement problem is solvable, with f-failure termination, in the model where process task time bounds of $[\ell_1, \ell_2]$ and bounds of d for all messages hold eventually, provided that $n > 2f$.

Outline

1. Failure Detection
2. Consensus
   - Problem Definition
   - Solution by Transformation of Synchronous Algorithms
   - PSynchAgreement
   - More Partially Synchronous Models
3. Further Reading

Further Reading

- Chapter 25, *Consensus With Partial Synchrony*, of Nancy Lynch's *Distributed Algorithms*.
- Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. Journal of the ACM, 41(1):122–152, January 1994. (PDF available at Nancy Lynch's web page at MIT.)
- Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. Journal of the ACM, 35(2):288–323, April 1988. (PDF available at Nancy Lynch's web page at MIT.)
- Flaviu Cristian, and Christof Fetzer, *The Timed Asynchronous Distributed System Model*, IEEE Transactions on Parallel and Distributed Systems, 10(6):642–657, June 1999 (PDF available from Christof Fetzer web page.)