Chapter 11

Expediting Decision in Synchronous Systems Prone to Process Crash Failures

The last section of the previous chapter showed that there is no synchronous round-based consensus (or interactive consistency) algorithm that can cope with t process crashes and allows the processes to always decide in less than $(t + 1)$ rounds (i.e., whatever the failure pattern).

This chapter focuses first on the case where less than t processes crash in an execution. It shows that the number of rounds can then be lowered to $\mathsf{min}(f + 2, t + 1)$ where f is the actual number of crashes ($0 \leq f \leq t$). The corresponding algorithm is based on a differential decision predicate involving the number of processes seen as crashed in the two last rounds.

The chapter presents also an *unbeatable* binary consensus algorithm, denoted CGM, where *unbeatability* means that its decision predicate cannot strictly be improved. More precisely, if there is an early deciding algorithm A based on a different decision predicate that improves the decision round with respect to CGM in a given execution, there is at least one execution of A in which a process strictly decides later than in CGM.

The chapter then presents the *condition-based* approach, which allows us to circumvent the $\mathsf{min}(f+2, t + 1)$ lower bound. It consists in restricting the allowable sets of input vectors. Finally, it is shown that enriching the round-based synchronous model $CSMP_{n,t}[\emptyset]$ with access to physical time and an appropriate *fast failure detector* allows decision to be to expedited.

Keywords Consensus, Early decision, Early stopping, Interactive consistency, Process crash, Round-based algorithm, Synchronous system.

11.1 Early Deciding and Stopping Interactive Consistency

Without loss of generality this section considers the interactive consistency agreement abstraction. The results trivially apply to consensus.

In the following, given an execution E, f denotes the number of processes that crash in E. Hence $0 \leq f \leq t$. While t is a parameter of the system model, and is known by the processes which can use its value in their local algorithms, no process knows the value of f when it starts executing.

11.1.1 Early Deciding vs Early Stopping

While $(t + 1)$ rounds are necessary (and sufficient) in worst case scenarios (Theorem 42), it might be supposed that, in executions where the number f of process crashes is small compared to the model

M. Raynal, *Fault-Tolerant Message-Passing Distributed Systems*,
https://doi.org/10.1007/978-3-319-94141-7_11

upper bound t, the number of rounds could be correspondingly small. This section shows that this is indeed the case. It presents a round-based algorithm which works in the model $CSMP_{n,t}[\emptyset]$ and where the processes decide in at most $\min(f+2, t+1)$ rounds. This is called *early decision*. Moreover, when a process decides, it stops its execution, which means that a process does not send messages after it has decided. This is called *early decision/stopping*.

A simple intuition for the $(f+2)$ (and not $(f+1)$) lower bound is the following. As there are only f failures in the considered execution, after $(f+1)$ rounds there is at least one process that executed a round in which it saw no failures. Thereby, this process knows which value can be decided, but, as $f \neq t$, it does not know if the other processes are aware of it. Hence, it needs an additional round to inform the other processes of this knowledge before deciding.

11.1.2 An Early Decision Predicate

From late decision to early decision Let us consider the non-early deciding interactive consistency algorithm described in Fig. 10.4. The aim is to modify it in order to obtain an early-deciding algorithm. This non-early deciding algorithm allows a process p_i not to send a message in a round r when p_i has not received new pairs $\langle k, v \rangle$ during the previous round $(r-1)$. As we have seen (Lemma 38), this does not prevent the processes that terminate round $(t+1)$ from having the very same vector of proposed values at the end of this round.

These "missing" messages can create a problem when we want a process p_i to decide "as early as possible". This is because, if p_i does not receive a message from process p_j during a round r, it cannot differentiate the case where p_j crashed from the case where p_j had nothing new to forward. To solve this problem, a process is required to follow these behavioral rules:

- A process broadcasts a message at every round until it decides or crashes.
- Any message indicates if its sender was about to decide after broadcasting it (during the same round).

These simple rules reduce the uncertainty on the state of p_j as perceived by p_i. Let r be the first round during which p_i does not receive a message from p_j. It follows from the previous rules that this message is missing either because p_j decided during round $r-1$, or because p_j crashed during $(r-1)$ (after it sent a message to p_i) or during round r (before it sent a message to p_i). Let us observe that, if p_j decided, it sent to p_i all the pairs $\langle k, v \rangle$ it previously received during the rounds r', $1 \leq r' \leq r-1$.

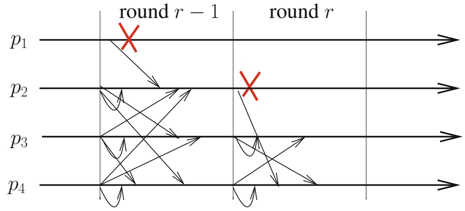
A predicate for early decision All that remains is to state a predicate that allows a process p_i to early decide by itself (i.e., before knowing that another process decided). Hence, assuming that no process decided up to round $(r-1)$, let us consider the following definitions:

- UP^r: the set of processes that start round r.
- R_i^r: the set of processes from which p_i received messages during round $r \geq 1$.
- R_i^0: the set of the n processes.

Let us notice that, while no process p_i knows the value of UP^r, it can compute the values of R_i^r and R_i^{r-1}. The following relation is an immediate consequence of (a) the previous definitions, (b) the previous sending rules, and (c) the fact that crashes are stable (no process recovers):

$$\forall\, r \geq 1: \quad R_i^r \subseteq UP^r \subseteq R_i^{r-1}.$$

Let us consider the particular case where, for p_i, two consecutive rounds $(r-1)$ and r are such that $R_i^r = R_i^{r-1}$. It follows from the previous relation that $R_i^r = UP_r = R_i^{r-1}$, which means that p_i received during round r a message from every process that was alive at the beginning of round r. This is illustrated in Fig. 11.1, where p_1 crashes during round $(r-1)$ and p_2 crashes during round r



$R_4^{r-1} = UP^r = R_4^r = \{2, 3, 4\}$

Figure 11.1: Early decision predicate

(this is indicated with crosses on p_1 and p_2 process axes). As far as messages are concerned, only the messages that are received by non crashed processes are indicated.

It follows that $R_i^r = R_i^{r-1}$ is the predicate we are looking for. It means that p_i received (during the rounds 1 to r) all the pairs $\langle k, v \rangle$ known by the processes that are alive at the beginning of r. To put it another way, all the other pairs $\langle \ell, w \rangle$ are lost forever and consequently no process can learn them in a future round. Process p_i can consequently decide the current value of its local vector $view_i$.

Inform the other processes before deciding It is not because the predicate $R_i^r = R_i^{r-1}$ is satisfied at process p_i, that $R_j^r = R_j^{r-1}$ is necessarily satisfied at another process p_j. As an example, when we consider the end of round r in Fig. 11.1, p_4 can be the only process that knows some pair $\langle k, v \rangle$ that has been forwarded only to p_1, which – before crashing – forwarded it only to p_2, which in turn – before crashing – forwarded it only to p_4. In this case, if p_4 decided during round r and stops executing just after deciding, it would decide a different vector from the vector decided by other processes.

This issue can be easily solved by directing p_i to execute an additional $(r+1)$ round during which it forwards the new pairs $\langle k, v \rangle$ it learned during round r. It also indicates in the corresponding message that its local early decision predicate was satisfied during round r. In this way, a process p_j that receives this message learns that the vector was decided by p_i. Hence, p_j learns that it can decide in the next round $(r+2)$, i.e., after having forwarded all the pairs $\langle k, v \rangle$ it learned from p_i during round $r(r+1)$.

11.1.3 An Early Deciding and Stopping Algorithm

The early deciding algorithm based on the previous design principles is described in Fig. 11.2. As indicated, this algorithm is obtained from the non-early deciding interactive consistency algorithm described in Fig. 10.4. In order to make it easier to understand, the lines with exactly the same statements are numbered the same way. The new lines are numbered N1 to N4, and the numbers of the two lines that are modified are prefixed by M.

Local data structures In addition to the vector $view_i[1..n]$ and the set variable new_i, a process manages three additional local variables: two Boolean variables and an array of integers.

- $nbr_i[0..n]$ is an array of integers comprised between 1 and n, such that $nbr_i[r]$ is the number of processes from which p_i received a message during round r, i.e., $nbr_i[r] = |R_i^r|$. By definition $nbr_i[0] = n$.

  As crashes are stable, the early decision predicate $R_i^{r-1} = R_i^r$ can be re-stated $nbr_i[r-1] = nbr_i[r]$. (As only nbr_i^{r-1} and nbr_i^r are needed, the array $nbr_i[0..n]$ can be trivially replaced by two local variables. This is not done here for clarity of the exposition.)
- $early_i$ is a Boolean initialized to `false`. It is set to `true` when the local early decision predicate is satisfied, or when p_i learns that another process is about to decide.

- $decide_i$ is a Boolean set to true when p_i receives a message from a process p_j indicating that $early_j$ is satisfied.

Let us remember that the macro-operation broadcast() is unreliable. If a process crashes during its invocation, an arbitrary subset of processes receive the message that has been broadcast.

Process behavior The lines that are modified with respect to the non-early deciding algorithm are line M1 and M4. The first concerns the initialization. The second concerns the addition of the current value of the Boolean $early_i$ to the message p_i broadcasts at every round.

As far as the new lines are concerned, we have the following. Line N2 gives its value to $nbr_i[r]$. At line N3, p_i sets $decide_i$ to $true$ if, and only if, it has received a round r message from a process p_j indicating that p_j is about to decide (i.e., $early_j$ is equal to $true$).

For the lines N1 and N4 let us first consider line N4. At that line, p_i sets $early_i$ to $true$ if, during the current round, its local early decision predicate has become true or p_i has received a round r message with $early_j = true$. To put it another way, $early_i$ is set to $true$ as soon as p_i learns (directly from its local predicate, or indirectly from another process) that it can early decide.

Let r be the first round at which $early_i$ becomes true. During round $(r+1)$ p_i broadcasts EST$(new_i, true)$ thereby indicating that it is about to early decide during that round. It then early decides (and stops) at line N1.

```
operation propose (v_i) is
(M1)  view_i ← [⊥,...,⊥]; view_i[i] ← v_i; new_i ← {⟨i, v_i⟩}; nbr_i[0] ← n; early_i ← false;
(2)   when r = 1, 2, ..., (t+1) do
(3)   begin synchronous round
(M4)     broadcast EST(new_i, early_i) end if;
(5)      for each j ∈ {1,...,n} \ {i} do
(6)         if (new_j received from p_j) then recfrom_i[j] ← new_j else recfrom_i[j] ← ∅ end if;
(7)      end for;
(N1)     if (early_i) then return(view_i) if;
(N2)     nbr_i[r] ← number of processes from which round r messages have been received;
(N3)     decide_i ← ⋁(early_j received during round r);
(8)      new_i ← ∅;
(9)      for each j such that (j ≠ i) ∧ (recfrom_i[j] ≠ ∅) do
(10)        foreach ⟨k, v⟩ ∈ recfrom_i[j] do
(11)           if (view_i[k] = ⊥) then view_i[k] ← v; new_i ← new_i ∪ {⟨k, v⟩} end if
(12)        end for
(13)     end for;
(N4)     if ((nbr_i[r−1] = nbr_i[r]) ∨ decide_i) then early_i ← true end if;
(14)     if (r = t+1) then return(view_i) end if
(15)  end synchronous round.
```

Figure 11.2: An early deciding t-resilient interactive consistency algorithm (code for p_i)

11.1.4 Correctness Proof

Let var_i^r denote the value of the local variable var_i at the end of round r. The sentence "p_i knows the pair $\langle k, v\rangle$" is a shortcut to say "$view_i[k] = v$". Process p_i "learned" this pair at round 0 if $i = k$, or at round $r > 0$ during which it receives for the first time a set new_j such that $\langle k, v\rangle \in new_j$.

Lemma 42. *If a process p_i decides at line* N1 *of round r, it knows all the pairs $\langle k, v\rangle$ known by the processes that had not crashed at the beginning of round $(r-1)$. Moreover, no more pairs can be learned by a process in a round $r' \geq r$.*

Proof If p_i decides at round r, it previously set $early_i$ to the value true at line N4 of round $(r-1)$. There are two cases.

- Case 1. $nbr_i[r-2] = nbr_i[r-1]$ at line N4 of round $(r-1)$. In this case, at every round r', $1 \le r' \le r-1$, p_i received a message from each process in R_i^{r-1}. Consequently, it knows all the pairs known by the processes in R_i^{r-1}. Moreover, as $nbr_i[r-2] = nbr_i[r-1]$, the set R_i^{r-1} is equal to UP^{r-1} (the set of processes alive at the beginning of round $(r-1)$). Hence, p_i knows all the pairs $\langle k, v\rangle$ known by the processes that had not crashed at the beginning of round $(r-1)$. Consequently no other pair can ever be known by a process in the future, which completes the proof of the lemma for this case.
- Case 2. $decide_i = \texttt{true}$ at line N4 of round $(r-1)$. In this case, there is a round $r' < r$ and a chain of distinct processes $p_{j1}, \ldots, p_{jx}$ ending at p_i such that (a) $nbr_{j1}[r'-1] = nbr_{j1}[r']$, and (b) p_{j1} sent $\text{EST}(-, \texttt{true})$ to p_{j2} during round $r'+1$, which in turn sent $\text{EST}(-, \texttt{true})$ to p_{j3} during round $r'+2$, etc., until p_{jx} that sent $\text{EST}(-, \texttt{true})$ to p_i during round $r-1$, and p_i consequently set $decide_i$ to `true` when it received that message.

  It follows from Case 1 that, at the end of round r', p_{j1} knew all the pairs known by the processes that had not crashed at the beginning of round r'. Hence, p_i knows all these pairs (at least from the chain of $\text{EST}(-, \texttt{true})$ messages starting at p_{j1} and ending at p_{jx}). Consequently, p_i knows all the pairs $\langle k, v\rangle$ known by the processes that had not crashed at the beginning of round r'. As no pair can be learned by a process in a later round, p_i knows all the pairs $\langle k, v\rangle$ known by the processes that had not crashed at the beginning of round $(r-1)$, which completes the proof of the lemma.

$\square_{Lemma\ 42}$

Lemma 43. *No two processes decide different vectors.*

Proof We consider three cases. Let p_i and p_j be two processes that decide.

- Case 1: no process decides at line N1. The proof is then exactly the same as the proof of the base non-early deciding algorithm (Lemma 38).
- Case 2: no process decides at line 14. The fact that $view_i^r = view_j^{r'}$ follows from Lemma 42.
- Case 3: some processes (e.g., p_i) decide at line N1 of a round r, while other processes (e.g., p_j) decide at line 14 of round $(r+1)$.

  Let us first observe that, in this case, $r = t$ or $r = t+1$. If p_i decided at line N1 of round $r < t$, the message $\text{EST}(-, \texttt{true})$ it broadcast at line 4M before deciding at line N1 was received during round r by p_j, which set $decide_j$ to `true` at line N3, entailing its decision at line 14 of round $(t+1)$ (case assumption). This is possible only if $r = t$ or $r = t+1$.

  It follows from Lemma 42 that p_i knows all the pairs that can be known at the beginning of round $(r-1)$. Moreover, from round 1 to round r, it transmitted all these pairs to p_j. It follows that $view_i^r = view_j^{t+1}$.

$\square_{Lemma\ 43}$

Theorem 43. *Let $1 \le t < n$. The algorithm described in Fig. 11.2 implements the* interactive consistency *agreement abstraction in $CSMP_{n,t}[\emptyset]$.*

Proof The ICC-Termination property is a direct consequence of the synchrony assumption of the model: no process executes more than $(t+1)$ rounds. The ICC-agreement property follows from Lemma 43. The proof of the ICC-validity property is the same as for the non-early deciding algorithm.

$\square_{Theorem\ 43}$

Theorem 44. *Let f denote the number of crashes in a given execution ($0 \le f \le t$). No process executes more than* $\min(f+2, t+1)$ *rounds.*

Proof As previously mentioned, the fact that a process executes at most $(t + 1)$ rounds follows from the text of the algorithm and the synchrony assumption. For the $(f + 2)$ rounds lower bound, let us consider two cases.

- Case 1. There is a process p_i that decides at line N1 of a round $d \leq f+1$. In this case, just before deciding at line N1 during round $(f + 1)$, p_i broadcast $\text{EST}(-, \texttt{true})$ at line 4M. It follows that each process p_j that terminates the round $(f + 1)$ receives the message $\text{EST}(-, \texttt{true})$ sent by p_i, and consequently updates $early_j$ to `true` during the round $(f + 1)$ (lines N3 and N4). It follows that, if p_j does not crash by the end of the round $(f + 2)$, it decides at line N1 of this round, which proves the theorem for this case.
- Case 2. No process decided by round $d = f + 1$. Let p_i be any process that terminates this round. As p_i did not decide by the end of round $(f + 1)$, we have $nbr_i[r' - 1] \neq nbr_i[r']$ for any round r', $1 \leq r' \leq f$. As there are exactly f crashes, it follows that we have:
  - $nbr_i[0] = n$, $nbr_i[1] = n - 1$, $nbr_i[2] = n - 2$, etc., $nbr_i[f - 1] = n - (f - 1)$ and $nbr_i[f] = n - f$ (there is one crash per round, and the process that crashes does not send a message to p_i), and
  - $nbr_i[f + 1] = n - f$.

  Consequently $nbr_i[f] - nbr_i[f + 1] = 0$. Hence, p_i sets $early_i$ to `true` at line N4 of the round $(f+1)$, and if it does not crash during the round $(f+2)$, it decides at line N1 of this round. Let us finally observe that, as p_i is any process that terminates round $(f + 1)$, the reasoning applies to all processes that execute round $(f + 2)$, which completes the proof of the theorem.

$\square_{Theorem\ 44}$

11.1.5 On Early Decision Predicates

Let $\mathsf{DIFF}(i, r)$ denote the previous early decision predicate (namely, $nbr_i[r] - nbr_i[i, 1] = 0$).

Another early detection predicate Let $faulty_i[r] = n - nbr_i[r]$, i.e., the number of processes that p_i perceives as crashed. The predicate $\mathsf{COUNT}(i, r) \equiv (faulty_i[r] < r)$ is another correct early decision predicate that can be used instead of $\mathsf{DIFF}(i, r)$. This is because $\mathsf{COUNT}(i, r)$ is satisfied at the first round r such that this round number is higher than the number of processes currently perceived as crashed by p_i. Put differently, from p_i's point of view, there are currently less crashed processes than the number of rounds it has executed, i.e., for p_i there is a round r', $1 \leq r' \leq r$, without crashes. Hence, at the end of this round, the vector $view_i$ contains the values v of all the pairs $\langle k, v\rangle$ that were known at the beginning of r', which means that no more pairs can be known by any process in the future.

The reader can check that the early-decision algorithm described in Fig. 11.2 works when, at line N4, the decision predicate $\mathsf{DIFF}(i, r) \equiv (nbr_i[r] - nbr_i[i, 1] = 0)$ is replaced by the predicate $\mathsf{COUNT}(i, r) \equiv (faulty_i[r] < r)$.

Comparing the predicates $\mathsf{COUNT}()$ and $\mathsf{DIFF}(i, r)$ Hence the question: While both $\mathsf{DIFF}(i, r)$ and $\mathsf{COUNT}(i, r)$ ensure that the processes decide in at most $\mathsf{min}(f + 2, t + 1)$ rounds in the worst cases, is one predicate better than the other? We show here that $\mathsf{DIFF}(i, r)$ is better than $\mathsf{COUNT}(i, r)$. To this end we prove the following theorem.

Theorem 45. *(a) Given an execution, let $r \geq 2$ be the first round at which $\mathsf{COUNT}(i, r)$ is satisfied. We have $\mathsf{COUNT}(i, r) \Rightarrow \mathsf{DIFF}(i, r)$.*
(b) Given an execution, let $r \geq 2$ be the first round at which $\mathsf{DIFF}(i, r)$ is satisfied. There are failure patterns for which $\mathsf{DIFF}(i, r) \wedge \neg\mathsf{COUNT}(i, r)$.

operation propose (v_i) **is**
$est_i \leftarrow v_i$; $nbr_i[0] \leftarrow n$; $early_i \leftarrow$ `false`;
when $r = 1, 2, \ldots, (t+1)$ **do**
begin synchronous round
    broadcast EST$(est_i, early_i)$;
    if $(early_i)$ **then** return (est_i) **end if**;
    let $nbr_i[r]$ = number of messages received by p_i during r;
    let $decide_i \leftarrow \bigvee(early_j$ values received during current round $r)$;
    $est_i \leftarrow \min(\{est_j$ values received during current round $r\})$;
    if $\big((nbr_i[r-1] = nbr_i[r]) \vee decide_i\big)$ **then** $early_i \leftarrow$ `true` **end if**
    if $(r = t+1)$ **then** return(est_i) **end if**
end synchronous round.

Figure 11.3: Early stopping synchronous consensus (code for p_i, $t < n$)

Proof Let us first prove item (a). As r is the first round during which $\mathsf{COUNT}(i, r) \equiv (faulty_i[r] < r)$ is satisfied, $\mathsf{COUNT}(i, r-1)$ is false, i.e., $faulty_i[r-1] \geq r-1$. It follows from $faulty_i[r] < r$ and $faulty_i[r-1] \geq r-1$ that $faulty_i[r] - faulty_i[r-1] < 1$., i.e., $(n - nbr_i[r]) - (n - nbr_i[r-1]) < 1$. Combined with the fact that $nbr_i[r-1] \geq nbr_i[r]$, we obtain $nbr_i[r] - nbr_i[r-1] = 0$, which concludes the proof of item (a).

Let us now prove item (b). To this end we exhibit a counter-example. Let us consider a run in which $2 \leq x \leq t$ processes crashed before taking any step, and then no other process crashes.

The predicate $\mathsf{COUNT}(i, r) \equiv (faulty_i[r] < r)$ becomes true for the first time at round $x+1$. Let us now look at the predicate $\mathsf{DIFF}(i, r) \equiv (nbr_i[r] - nbr_i[r-1] = 0)$. We have $nbr_i[1] = nbr_i[2] = n - x$. Consequently, $\mathsf{DIFF}(i, 2)$ is satisfied. As $x \geq 2$, it follows that $\neg\mathsf{COUNT}(i, 2) \wedge \mathsf{DIFF}(i, 2)$, which concludes the proof of item (b). $\Box_{Theorem\ 45}$

Discussion The previous theorem shows that, while both the early decision predicates $\mathsf{DIFF}(i, r)$ and $\mathsf{COUNT}(i, r)$ allow the processes to decide and stop by round $r = \min(f+2, t+1)$, the predicate $\mathsf{DIFF}(i, r) \equiv (nbr_i[r] - nbr_i[r-1] = 0)$ is better than the predicate $\mathsf{COUNT}(i, r) \equiv (faulty_i[r] = n - nbr_i[r])$, in the sense that there are failure patterns for which $\mathsf{DIFF}(i, r)$ allows the processes to terminate before round $r = \min(f+2, t+1)$.

This is due to the fact that $\mathsf{DIFF}(i, r)$ is a *differential predicate*: it takes into consideration the actual failure pattern, namely, a process computes the number of process crashes it perceives during a round (the value of this number is $nbr_i[r] - nbr_i[r-1]$). Whereas the predicate $\mathsf{COUNT}(i, r)$ is based only on the number of processes perceived as crashed by p_i since the beginning of the execution. This means that, whatever the actual failure pattern, $\mathsf{COUNT}(i, r)$ always considers the worst case scenario in which there is one crash per round. However, when using $\mathsf{DIFF}(i, r)$, the fact that crashes occur in the very same round is taken into account and allows for a faster decision.

As an example, let us consider the case where no process crashes. The algorithm with the predicate $\mathsf{DIFF}(i, r) \equiv (nbr_i[r] - nbr_i[r-1] = 0)$ allows each process to decide and stop in two rounds, whatever the value of t. If any number of processes crash initially (i.e., before the algorithm starts), and later no more process crashes, it allows the correct processes to decide in three rounds.

11.1.6 Early Deciding and Stopping Consensus

The algorithm described in Fig. 11.3 describes an early deciding and stopping consensus algorithm. This algorithm, where a process decides the smallest value it has ever seen is directly obtained from the interactive consistency early-deciding algorithm described in Fig. 11.2. Its proof is left to the reader.

11.2 An Unbeatable Binary Consensus Algorithm

The notion of an unbeatable predicate for early deciding/stopping consensus algorithms in the model $CSMP_{n,t}[\emptyset]$ is due A. Castañeda, Y. Gonczarowski, and Y. Moses (2014). This notion is based on knowledge theory. The associated binary consensus algorithm CGM, which is presented in this section, is also due to the same authors.

11.2.1 A Knowledge-Based Unbeatable Predicate

Underlying intuition The idea is to allow processes to decide as soon as possible on a preferred value (let us consider 0). The other value (1) can be decided by a process only when it is sure that no process can decide on the preferred value 0. More operationally, we have the following:

- A process p_i can safely decide on 0 as soon as it knows that every correct process knows that the value 0 was proposed. This occurs when p_i knows that each correct process received a message indicating some process proposed 0.
- A process p_i can safely decide on 1 as soon as it knows that no active process received a message indicating a process proposed 0. In this case, if it was initially present, 0 disappeared from the system.

The knowledge-based predicate $\mathsf{PREF0}$ Given an execution, we use the following terminology:

- "A process p_j is $revealed$ to process p_i in a round r" if either p_i knows all the values known by p_j at the beginning of r, or p_i knows that p_j crashed before round r. Hence, if, in round r, p_j is $revealed$ to p_i, it cannot broadcast values not yet know by p_i.
- "A round r is $revealed$ to process p_i" if every process p_j is revealed to p_i in round r. When this occurs, p_i knows all the values that are in the system at the beginning of round r.

The knowledge-based predicate $\mathsf{PREF0}$, used to decide 0 as soon as possible, is defined as follows:

$$\mathsf{PREF0} \stackrel{def}{=} \mathsf{correct0}(i, r) \vee \mathsf{revealed0}(i, r)$$

where

- $\mathsf{correct0}(i, r)$ denotes the predicate "p_i knows that at least one correct process knows in round r that 0 was proposed", and
- $\mathsf{revealed0}(i, r)$ denotes the predicate "a round $r' \leq r$ has been revealed to p_i".

Let us notice that, if $\mathsf{correct0}(i, r)$ holds, all correct processes will know 0 was proposed by the end of round $(r + 1)$.

An example illustrating the predicate $\mathsf{correct0}(i, r)$ Let us consider a process p_i, whose proposed value is 0, which, during the first round, broadcasts it and receives messages from the other processes. Hence, at the end of the first round, it knows that every alive process knows the value 0 was proposed. Therefore, the predicate $\mathsf{correct0}(i, 1)$ is satisfied, and (if it does not crash) p_i can decide on 0 at the of the first round. Moreover, this is independent of the possible crash of the other processes.

Let p_j be a process p_j which proposes value 1. According to the failure pattern, it can be the only process that received the value 0 from p_i; hence, $\mathsf{correct0}(j, 1)$ does not hold, and it cannot decide 0 in this round. Moreover, p_j is prevented from deciding 1 because it knows 0 was proposed.

The reader can check that this scenario is not restricted to the first round, and, according to the failure pattern, can occur at any round r.

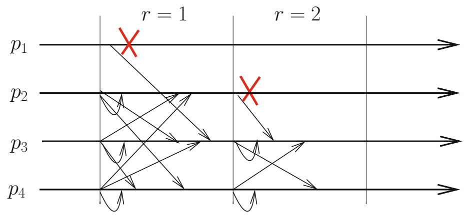


Figure 11.4: The early decision predicate $\mathsf{revealed0}(i, r)$ in action

An example illustrating the predicate $\mathsf{revealed0}(i, r)$ Let us consider an execution involving four processes which all propose value 1, and where the failure and message pattern is as depicted in Fig. 11.4.

During the first round, p_4 receives a message from p_2 and p_4 but not from p_1. Hence, it knows that p_1 crashed, but it does not know the value proposed by p_1 nor whether it sent its value to p_2 and p_3 before crashing. Actually, before crashing, p_1 sent its value to p_3 only. During the second round, p_4 receives a message from p_3 (hence it learns that p_1 proposed 1), but does not receive a message from p_2, which crashed after sending a message to p_3.

Despite the fact that it sees a crash at every round, p_4 knows, during the second round, that only the value 1 has been proposed. Hence, $\mathsf{revealed0}(4, 2)$ is satisfied. Consequently, p_4 can safely decide 1. It is easy to see that the local predicate $\mathsf{revealed0}(3, 1)$ is also satisfied.

11.2.2 $\mathsf{PREF0}()$ with Respect to $\mathsf{DIFF}()$

Theorem 45 showed that the predicate $\mathsf{DIFF}(i, r)$ is strictly stronger than $\mathsf{COUNT}(i, r)$. The next theorem shows that (assuming an algorithm in which, at every round, each process broadcasts everything it knows) $\mathsf{PREF0}()$ is strictly stronger than $\mathsf{DIFF}()$.

Theorem 46. *(a) Given an execution, let r be the first round at which* $\mathsf{PREF0}(i, r)$ *is satisfied. We have* $\mathsf{DIFF}(i, r) \Rightarrow \mathsf{PREF0}(i, r)$.
(b) Given an execution, let r be the first round at which $\mathsf{DIFF}(i, r)$ *is satisfied. There are failure patterns for which* $\mathsf{PREF0}(i, r) \wedge \neg\mathsf{DIFF}(i, r)$.

Proof Let us first prove item (a). Since $\mathsf{DIFF}(i, r)$ is satisfied, we have $nbr_i[r-1] = nbr_i[r]$. Therefore, in round r, p_i receives a message from any process p_j that sends a message to p_i in round $r-1$. Moreover, p_i knows that all other processes crash before round r simply because it does not get any message from them in round $(r-1)$. We conclude that round r is revealed to p_i, and the predicate $\mathsf{revealed}(i, r)$ holds. Consequently, $\mathsf{PREF0}(i, r)$ is satisfied.

To prove item (b), let us consider any execution in which (1) all processes propose 0, (2) p_n crashes without communicating its input to any process, and (3) all other processes are correct. Then, for every process p_i, $1 \le i \le n-1$, $\mathsf{revealed}(i, 1)$ is true, as p_i proposes 0 and sends it to every other process. Thus, $\mathsf{PREF0}(i, 1)$ is satisfied. In contrast, $\mathsf{DIFF}(i, r)$ is not satisfied because, as p_i does not receive a message from p_n, we have $nbr_i[0] = n \wedge nbr_i[1] = n-1$. $\Box_{Theorem\ 46}$

11.2.3 An Algorithm Based on the Predicate $\mathsf{PREF0}()$: CGM

As already indicated this binary consensus algorithm, which works in the model $CSMP_{n,t}[\emptyset]$, is due A. Castañeda, Y. Gonczarowski, and Y. Moses (2014).

Local variables Each process p_i manages the following local variables:

- $vals_i$: the set of proposed values known by p_i. It initially contains the value v_i proposed by p_i.
- $knew0_i$: a Boolean indicating that $0 \in vals_i$ at the end of the previous round.
- $correct0_i$: a Boolean indicating the predicate $\mathsf{correct0}(i, r)$ is satisfied in the current round r.
- $revealed_i$: a Boolean indicating the predicate $\mathsf{revealed}(i, r)$ is satisfied in the current round r.
- lg_i: a local directed graph whose vertices are pairs $\langle$process id, round number$\rangle$. The function $vertices(lg_i)$ (resp., $edges(lg_i)$) returns its current set of vertices (resp., edges).

  Initially this graph contains only the pair $\langle i, 0\rangle$. It is then enriched at every round r according to the messages received by p_i during round r.

Management of the local graphs lg_i The algorithm is a *full-information* algorithm. This means each process p_i sends its local state to all other processes at every round. It then follows that the local graph lg_i includes all the causal message paths that p_i can know until the current round.

There is a directed edge from the vertex $\langle j, r\rangle$ to the vertex $\langle k, r+1\rangle$ if p_i knows that p_k received a message from p_j in round $(r+1)$. As just mentioned, this message carries the local state of p_j at the end of round r. The relevant part of the local state of a process p_j (i.e., the part that is transmitted) is composed of its local variables $vals_i$ and lg_i.

Considering the execution depicted in Fig. 11.4, the next figures presents the values of the local graphs at the end of the rounds $r = 1$ (Fig. 11.5) and $r = 2$ (Fig. 11.6). (So not to overload the figure, the tips of the arrows are not depicted on the graphs.)

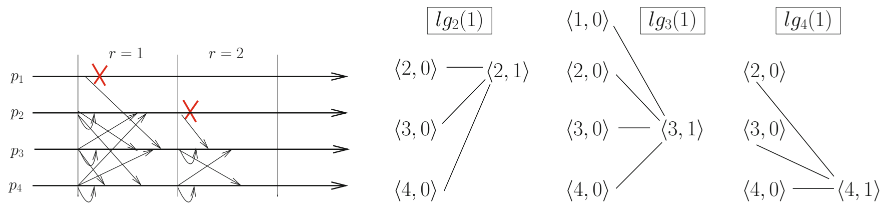


Figure 11.5: Local graphs of p_2, p_3, and p_4 at the end of round $r = 1$

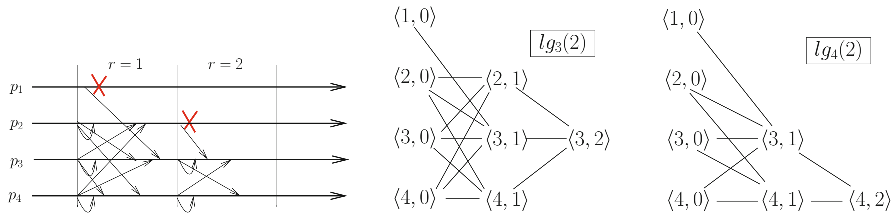


Figure 11.6: Local graphs of p_3 and p_4 at the end of round $r = 2$

Part 1 of the algorithm: communication and local state update This part is composed of the lines 5 and 7-11. When it starts a new round r, a process p_i sends its current local state to all processes, namely, the pair composed of $vals_i$ and its local knowledge (saved in its local graph lg_i) of the message exchanges that occurred up to the previous round (line 5). If its local flag $early_i$ is true, p_i early decides the value 0 (line 6). If $early$ is false and the value 0 was in the set $vals_i$ at the end of the previous round, p_i sets $knew0_i$ to `true`. This is because, as p_i just broadcast $vals_i$ (line 5),

```
operation propose(v_i) is
(1)   vals_i ← {v_i}; lg_i ← ({⟨i,0⟩}, ∅);
(2)   early_i, knew0_i, correct0_i, revealed_i ← false;
(3)   when r = 1, 2, ..., (t+1) do
(4)   begin synchronous round
(5)     broadcast MY_STATE(vals_i, lg_i);
(6)     if (early_i) then return(0) end if;
(7)     if (0 ∈ vals_i) then knew0_i ← true end if;
(8)     vals_i ← ⋃(vals_j values received during round r);
(9)     let n0_i = number of messages received in round r with 0 ∈ vals_j;
(10)    let nf_i = number of processes from which no message was received in round r;
(11)    lg_i ← ⋃(lg_j graphs received during round r and directed edges (⟨j,r−1⟩,⟨i,r⟩));
        % Testing correct0(i,r)
(12)    if (0 ∈ vals_i ∧ (knew0_i ∨ (t − nf_i ≤ n0_i))) then correct0_i ← true end if;
        % Testing revealed(i,r)
(13)    if (∃ r' ≤ r : ∀p_j :   (⟨j,r'⟩ ∈ vertices(lg_i))
                 ∨(∃⟨ℓ,r'⟩ ∈ vertices(lg_i) : (⟨j,r'−1⟩,⟨ℓ,r'⟩) ∉ edges(lg_i)))
(14)       then revealed_i ← true
(15)    end if;
        % Testing PREF0(i,r)
(16)    if (correct0_i) then return(0) end if;
(17)    if (revealed_i ∧ 0 ∉ vals_i) then return(1) end if;
(18)    if (revealed_i ∧ 0 ∈ vals_i) then early_i ← true end if
(19)  end synchronous round.
```

Figure 11.7: CGM: Early deciding synchronous consensus based on PREF0() (code for p_i, $t < n$)

and this set contains 0, it knows that all non-crashed processes receive its set $vals_i$ during the current round, and consequently knows 0 was proposed.

Process p_i then updates its local state ($vals_i$, $n0_i$, nf_i, lg_i) according to the values it has received and the number of processes from which it received them during the current round (lines 8-11).

Let us observe that, at line-11, the local graph lg_i is enriched as depicted in Fig. 11.5 and 11.6. In addition to the union of the graph lg_j, p_i adds the edge $\langle j, r-1\rangle, \langle i, r\rangle$ for each p_j from which it received a message during round r. Hence, once updated at line 11 of round r, lg_i implicitly contains all causal message chains ending at the vertex $\langle i, r\rangle$.

Part 2 of the algorithm: trying to progress to a decision This part is composed of lines 12-18 in which p_i computes correct0(i, r) and revealed(i, r) to expedite the decision (lines 16-18). This part is made up of three sets of statements.

- Process p_i first computes correct0(i, r) (line 12). There are two cases.
  - Case 1: $0 \in vals_i$ and $knew0_i$ = `true`. In this case, p_i knows that all non-crashed processes know the value 0 was proposed. This is because p_i sent it to them in its last message MY_STATE$(vals_i, lg_i)$. The predicate correct0(i, r) is then satisfied, and accordingly p_i sets *correct*0_i to `true`.
  - Case 2: $0 \in vals_i$ and $knew0_i$ = `false`. In this case, p_i learned 0 was proposed in the current round. If $t - nf_i \leq n0_i$, during the current round r, at least $(n - nf_i + 1)$ processes know 0 was proposed. (The "+1" comes from the process p_i itself, which during the current round learned 0 is a proposed value.) As at most $(n - nf_i)$ processes may crash, it follows that at least one correct process knows 0 was proposed. Consequently, the predicate correct0(i, r) is satisfied, and p_i sets *correct*0_i to `true`.

- Then, process p_i computes revealed(i, r) (lines 13-14).
  This predicate is true if a round $r' \leq r$ has been revealed to p_i, where "a round r' is revealed to

p_i" if p_i knows what was known by p_j at the beginning of round r', or p_j crashed before round r'. This is captured by the predicate of line 13:

$$\exists\, r' \leq r : \forall\, p_j :$$
$$\big(\langle j, r'\rangle \in vertices(lg_i)\big) \vee \big(\exists\, \langle \ell, r'\rangle \in vertices(lg_i) : (\langle j, r'-1\rangle, \langle \ell, r'\rangle) \notin edges(lg_i)\big).$$

Process p_i verifies on lg_i if a round is revealed to it, namely, if there is a round $r' \leq r$ such that, for each process p_j, we have:

- a causal chain of messages from the vertex $\langle j, r'\rangle$ (p_j at the beginning of $r'+1$) to $\langle i, r\rangle$ (p_i at the end of r), which amounts to check $\langle j, r'\rangle \in vertices(lg_i)$, or
- a vertex $\langle \ell, r'\rangle \in vertices(lg_i)$, such that $(\langle j, r'-1\rangle, \langle \ell, r'\rangle) \notin edges(lg_i)$ (p_ℓ did not receive a message from p_j in round r', hence p_j crashed).

- Finally, p_i strives to entail an early decision (lines 16-18).
  - If $\mathsf{correct0}(i, r)$ is satisfied, it decides 0 (line 16).
  - If $\mathsf{correct0}(i, r)$ is not satisfied, $0 \notin vals_i$, but $\mathsf{revealed}(i, r)$ is satisfied (line 17), it safely decides 1 (round r is revealed and no non-crashed process saw 0).
  - Finally, if $\mathsf{correct0}(i, r)$ is not satisfied, $\mathsf{revealed}(i, r)$ is satisfied, and $0 \in vals_i$, p_i sets early to `true` (line 18), and proceeds to the next round. During the round $(r+1)$, it broadcasts $vals_i \ni 0$ (to inform all other processes on the 0 proposal), and decides (line 6).

Theorem 47. *Let $1 \leq t < n$. The algorithm described in Fig. 11.7 implements the* binary consensus *agreement abstraction in $CSMP_{n,t}[\emptyset]$. Moreover, a process executes at most* $\mathsf{min}(f+2, t+1)$ *rounds.*

Proof (Sketch) The CC-termination property follows from the synchrony property of the model (the progress of rounds is due to the model). The CC-validity property follows from the updates of $vals_i$, line 12, and lines 16-18.

CC-agreement property follows from the observation that the only way for a process to decide 1 is to be sure that no process will ever know the value 0 was proposed. The formalization of this argument is the topic of Exercise 2 of Section 11.7.

The lower bound on the number of rounds is an immediate consequence of Theorem 46 and Theorem 44. $\square_{Theorem\ 47}$

11.2.4 On the Unbeatability of the Predicate PREF0()

As already indicated, PREF0() is unbeatable in the sense that it cannot *strictly* be improved. It is possible that there are early deciding predicates that improve the deciding round of a process in a given execution, but the deciding round of the same or another process in the same or another execution is then strictly worse.

An example is the predicate PREF1(), which is the same as PREF0() except the roles of 0 and 1 are exchanged. Its aim is to decide 1 as soon as possible. In the executions where all processes propose 0, PREF0() is fast, whatever the failure pattern, while PREF1() might need up to $(t+1)$ rounds. And vice versa, in the executions where all processes propose 1, PREF1() is fast, while PREF0() might need up to $(t+1)$ rounds.

11.3 The Synchronous Condition-based Approach

11.3.1 The Condition-based Approach in Synchronous Systems

An input vector $I[1..n]$ is a vector with one entry per process, such that $I[i]$ contains the value v_i proposed by process p_i. Let us remember that, in a synchronous system prone to process crash failures

($CSMP_{n,t}[\emptyset]$), both consensus and interactive consistency can be solved whatever the actual input vector and the value of the model parameter t, i.e., $0 \leq t < n$.

The underlying idea The condition-based approach is due to A. Mostéfaoui, S. Rajsbaum, and M. Raynal (2003). Its underlying idea is motivated by the following question: Is it possible to characterize sets of input vectors for which the processes always decide in less than $(t+1)$ rounds whatever the failure pattern? This section shows that the answer to this question is "yes". To this end, it first defines the notion of legal conditions and then presents a corresponding condition-based algorithm.

Definition of a condition A condition is a set of input vectors. Let $\mathcal{C}[x]$, $0 \leq x \leq t$, be the set (also called class) of conditions that allows consensus to be solved in at most $f_t(x)$ rounds, where $f_t(x) \leq t+1$ and $f_t(x+1) < f_t(x)$. The parameter x is called the *degree* of the class, and (by a slight abuse of language) we also say that it is the degree of the conditions C that are in $\mathcal{C}[x]$, i.e., $C \in \mathcal{C}[x]$ and $C \notin \mathcal{C}[y]$ where $y > x$. Section 11.3.2 shows that the classes $\{\mathcal{C}[x]\}_{0 \leq x \leq t}$ define the following hierarchy (Fig. 11.8), where $\mathcal{C}[0]$ contains the condition including all possible input vectors.

$$\mathcal{C}[t] \subset \mathcal{C}[t-1] \subset \cdots \subset \mathcal{C}[x] \subset \cdots \subset \mathcal{C}[1] \subset \mathcal{C}[0].$$

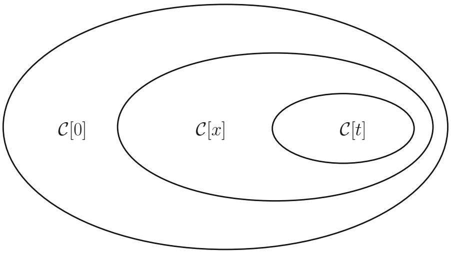


Figure 11.8: Hierarchy of classes of conditions

Section 11.3.5 will present a consensus algorithm that, when instantiated with a condition $C \in \mathcal{C}[x]$, allows the processes to decide in at most $f_t(x) = t+1-x$ rounds whatever (a) the actual input vector $I \in C$, and (b) the failure pattern.

This means that, if the condition C the algorithm is instantiated with belongs to $\mathcal{C}[t]$, the processes decide in one round (which is clearly optimal, when the decided value is not fixed a priori). At the other extreme, if the condition C the algorithm is instantiated with is the condition including all possible input vectors, the processes decide in at most $(t+1)$ rounds. Hence, there is a tradeoff between the number of input vectors of a condition C (as measured by its degree x) and the maximal number of rounds needed to decide.

11.3.2 Legality and Maximality of a Condition

Not any set C of input vectors allows the processes to decide in less than $(t+1)$ rounds whatever the pattern of up to t process crashes and the input vector $I \in C$. The notion of legality is introduced to capture the conditions that allow consensus to be solved in $(t+1-x)$ rounds.

Notations

- $\mathcal{V}$ denotes the set of values that can be proposed.
- $\mathsf{equal}(a, I)$ denotes the number of occurrences of the value a in the input vector I.

- $\mathsf{dist}(I1, I2)$ denotes the Hamming distance between the vectors $I1$ and $I2$ (the number of entries in which they differ).

Legality A condition C is x-*legal* if there is a function $h : C \mapsto \mathcal{V}$ with the following properties:

- $\forall\, I \in C : \#_{h(I)}(I) > x$,
- $\forall\, I1, I2 \in C : \big(h(I1) \neq h(I2)\big) \Rightarrow \big(\mathsf{dist}(I1, I2) > x\big)$.

The intuition that underlies this definition is the following. Given a condition C, each of its input vectors I allows a proposed value to be selected in order to be the value decided by the processes. That value is extracted from an input vector by the function $h()$, namely $h(I)$ is the value decided from input vector I.

To this end, $h()$ and all vectors I of C have to satisfy some constraints. The first constraint states that the value that the processes have to decide from I (this value is $h(I)$) has to be present enough in vector I. “Enough” means “more than x times”. This is captured by the first constraint defining x-legality: $\forall\, I \in C : \#_{h(I)}(I) > x$.

The second constraint states that, if different values are decided from different vectors $I1, I2 \in C$, then $I1$ and $I2$ must be “far apart enough” from one another. This is to prevent processes that would obtain different views of the input vector from deciding differently. This is captured by the second constraint defining x-legality: $\forall\, I1, I2 \in C : \big(h(I1) \neq h(I2)\big) \Rightarrow \big(\mathsf{dist}(I1, I2) > x\big)$.

The set of all x-legal conditions defines the class $\mathcal{C}[x]$. Hence, a set C of input vectors for which there is no function $h()$ as defined previously does not define a legal condition, and consequently $C \notin \mathcal{C}[x]$. Section 11.3.5 will describe a consensus algorithm that, when instantiated with the function $h()$ of a condition $C \in \mathcal{C}[x]$, allows the processes to decide in at most $(t + 1 - x)$ rounds whatever the input vector $I \in C$.

A relation with error-correcting codes The notion of a legal condition shows that there is a strong connection relating the consensus agreement abstraction and error-correcting codes: each input vector I encodes a value, namely the value that has to be decided from I. In this sense an input vector can be seen as a codeword. Given an upper bound d on the number of rounds we want to execute, the condition-based approach allows us to characterize which are the sets of input vectors (codewords) that allow consensus to be implemented in at most d rounds (where $d = t + 1 - x$). It is the set of conditions belonging to $\mathcal{C}[x]$. The condition-based approach thereby establishes a strong relation between agreement problems encountered in distributed computing and error-correcting codes.

The legal conditions C^x_{max} and C^x_{min} Assuming that the values that can be proposed can be totally ordered, a natural example of an x-legal condition is the one that favors the largest value present in an input vector. Let us call C^x_{max} this condition for a given degree x. Moreover, let $\mathsf{max}[I]$ denote the greatest value in the input vector I. C^x_{max} is defined as follows:

$$C^x_{max} \stackrel{def}{=} \{I \mid \mathsf{equal}(a, I) > x \text{ where } a = \mathsf{max}(I)\}.$$

Theorem 48. *The condition C^x_{max} is x-legal.*

Proof Let $\mathsf{max}(I)$ be the associated decision function $h()$. Due to the definition of C^x_{max}, the function $\max()$ trivially satisfies the first item of the definition of x-legality. Hence, we have only to show that $(\mathsf{max}(I1) \neq \mathsf{max}(I2)) \Rightarrow (\mathsf{dist}(I1, I2) > x)$ for any pair of vectors $I1, I2 \in C^x_{max}$.

Let $a = \mathsf{max}(I1)$ and $b = \mathsf{max}(I2)$. As a and b are different, one is greater than the other. Without loss of generality, let us assume $a > b$. As $b = \mathsf{max}(I2)$, we conclude that a does not appear in $I2$. As a appears more than x times in $I1$, it immediately follows that $\mathsf{dist}(I1, I2) > x$, which concludes the proof of the theorem. $\Box_{Theorem\ 48}$

Another natural example of an x-legal condition is the condition denoted C^x_{min} that favors the smallest value present in an input vector.

The legal condition C^x_{first} Another example is the condition that favors the most frequent value in an input vector. Let $\mathsf{first}(I)$ and $\mathsf{second}(I)$ be the values that appear the most frequently and the second most frequently in the input vector I, respectively. (If two values are equally frequent, we have $\mathsf{first}(I) = \mathsf{second}(I)$; a vector I made up of a single value is such that $\mathsf{first}(I) = n$ and $\mathsf{second}(I) = 0$.) The condition C^x_{first} defined as follows:

$$C^x_{first} \stackrel{def}{=} \{I \mid \mathsf{equal}(a, I) - \#_b(I) > x \text{ where } a = \mathsf{first}(I) \text{ and } b = \mathsf{second}(I)\}$$

is x-legal. The associated function $h()$ is the function $\mathsf{first}()$.

Maximal legal conditions An x-legal condition C is *maximal* if adding a vector to C makes it not x-legal. More formally, C is maximal if $C \cup \{I\}$ is not x-legal when $I \notin C$. The conditions C^x_{max} and C^x_{min} are maximal x-legal conditions, while C^x_{first} is x-legal but not maximal.

Illustrating the previous legal conditions C^x_{max} and C^x_{first} Let us consider a system of $n = 4$ processes, where up to $t = 3$ can crash. Table 11.1 presents the conditions C^x_{max} and C^x_{first} for $0 \le x \le t = 3$. The symbol "$\in$" means that the vector on the same line belongs to the condition defined by the corresponding column.

Input vector	C^0_{max}	C^1_{max}	C^2_{max}	C^3_{max}	C^0_{first}	C^1_{first}	C^2_{first}	C^3_{first}
$[0,0,0,0]$	$\in$	$\in$	$\in$	$\in$	$\in$	$\in$	$\in$	$\in$
$[0,0,0,1]$	$\in$				$\in$	$\in$		
$[0,0,1,0]$	$\in$				$\in$	$\in$		
$[0,0,1,1]$	$\in$	$\in$						
$[0,1,0,0]$	$\in$				$\in$	$\in$		
$[0,1,0,1]$	$\in$	$\in$						
$[0,1,1,0]$	$\in$	$\in$						
$[0,1,1,1]$	$\in$	$\in$	$\in$		$\in$	$\in$		
$[1,0,0,0]$	$\in$				$\in$	$\in$		
$[1,0,0,1]$	$\in$	$\in$						
$[1,0,1,0]$	$\in$	$\in$						
$[1,0,1,1]$	$\in$	$\in$	$\in$		$\in$	$\in$		
$[1,1,0,0]$	$\in$	$\in$						
$[1,1,0,1]$	$\in$	$\in$	$\in$		$\in$	$\in$		
$[1,1,1,0]$	$\in$	$\in$	$\in$		$\in$	$\in$		
$[1,1,1,1]$	$\in$	$\in$	$\in$	$\in$	$\in$	$\in$	$\in$	$\in$

Table 11.1: Examples of (maximal and non-maximal) legal conditions

11.3.3 Hierarchy of Legal Conditions

It is easy to see that C^{x+1}_{max} contains C^x_{max} while C^x_{max} does not contain C^{x+1}_{max}. Hence, $C^t_{max} \subset C^{t-1}_{max} \cdots \subset C^x_{max} \cdots \subset C^0_{max}$. As $\forall\, x, 0 \le x \le t$, $C^x_{max} \in \mathcal{C}[x]$, it follows (as previously mentioned) that the classes $\{\mathcal{C}[x]\}_{0 \le x \le t}$ define a strict hierarchy, depicted in Fig. 11.8.

11.3.4 Local View of an Input Vector

Let I be an input vector of an x-legal condition C. A *view* J of I (denoted $J \leq I$) is a vector that is identical to I except that at most x entries can be equal to $\perp$.

From an operational perspective, a view captures the non-$\perp$ entries of an input vector that a process obtains by receiving messages.

Lemma 44. *Let C be an x-legal condition and $I1$ and $I2$ two input vectors of C. If there is a view J such that $J \leq I1$ and $J \leq I2$, we have $h(I1) = h(I2)$.*

Proof Let us assume by contradiction that there is an x-legal condition C that has two vectors $I1$ and $I2$ such that (a) there is a view $J \leq I1$ and $J \leq I2$, and (b) $h(I1) \neq h(I2)$.

As $J \leq I1$ and $J \leq I2$, we have $\mathsf{dist}(J, I1) \leq x$ and $\mathsf{dist}(J, I2) \leq x$. From these inequalities, the fact that J has at most x entries equal to $\perp$, and the fact that the entries of J that differ in $I1$ or $I2$ are its only entries equal to $\perp$, it follows that $\mathsf{dist}(I1, I2) \leq x$.

However, as $h(I1) \neq h(I2)$, it follows from the second item of the definition of x-legality of C, that $\mathsf{dist}(I1, I2) > x$, which contradicts the previous observation, and concludes the proof. $\square_{Lemma\ 44}$

The previous lemma allows the definition of the selection function $h()$ associated with an x-legal condition C to be extended to views as follows.

Extending to views the definition of the function $h()$ If I is an input vector of an x-legal condition C, and J is a view of I, then the function $h()$ is extended as follows $h(J) = h(I)$.

11.3.5 A Synchronous Condition-based Consensus Algorithm

A condition-based consensus algorithm is presented in Figure 11.9. The parameter x is the degree of the condition C the algorithm is instantiated with. The function $h()$ is the selection function associated with this x-legal condition.

Local variables In addition to the local variable $view_i$ (whose meaning is similar to the one of the same variable used in the previous algorithm), a process p_i manages two local variables, both initialized to the default value $\perp$. This default value is assumed to be smaller than any value that can be proposed by a process.

- The aim of v_cond_i is to keep (once known) the value $h(I)$ decided from the input vector I.
- The aim of v_tmf_i is to contain the value that will be decided when (as we will see below) it is not possible to use the function $h()$ to decide a value from the input vector. (v_tmf stands for *too many failures*.)

Process behavior The behavior of p_i depends on the round.

- During the first round, a process p_i broadcasts the value it proposes (message EST1(v_i) sent at line 4), and builds its local view of the input vector during the receive phase (line 5). Then, p_i counts the number of entries of its view that are equal to $\perp$. There are two cases.
  - If $\mathsf{equal}(\perp, view_i) \leq x$ (line 6), p_i knows enough entries of the input vector in order to use the selection function $h()$ associated with the x-legal condition the algorithm is instantiated with. In that case, p_i computes $h(view_i)$ and saves it in v_cond_i.

- If equal$(\perp, view_i) > x$ (line 7), there are too many failures for $h()$ to be used. This is because, in order to be known before being decided, a value must be present at least once in a local view of the input vector. Hence, when more than x entries of the local view of p_i are equal to $\perp$, $h()$ is meaningless. In this case, p_i behaves as in a classic consensus algorithm. It computes the greatest proposed value it knows and saves it in v_tmf_i.

  The case of an x-legal condition such that $x = t$ is particular. This is because, if $x = t$, we necessarily have equal$(\perp, view_j) \leq x$ at any process that does not crash by the end of the first round. Consequently, no process p_j needs more rounds to know the value decided from the condition. It follows that any p_j can safely decide $h(view_j)$ during the very first round (line 9).

- From round 2 until round $(t + 1 - x)$, p_i first broadcasts its current state (with the message EST2(v_cond_i, v_tmf_i), line 13), then it early decides the value of v_cond_i, if it is not equal to $\perp$ (line 14). Let us observe that, in this case, v_cond_i was different from $\perp$ at the end of the previous round, and consequently, its value is carried by the message EST2() that p_i has just broadcast.

  If $v\_cond_i = \perp$, p_i updates it to the value decided from the condition if it has received such a value from another process (line 15). It also updates the value of v_tmf_i in case no value can be computed from the condition (line 16).

  Finally, if $r = t + 1 - x$, p_i decides (line 18). The decided value is the non-$\perp$ value kept in v_cond_i if there is one. Otherwise, it is the value kept in v_tmf_i.

operation propose$_x$ (v_i) **is**
(1) $view_i \leftarrow [\perp, \ldots, \perp]$; $view_i[i] \leftarrow v_i$; $v\_cond \leftarrow \perp$; $v\_tmf_i \leftarrow \perp$;
(2) **when** $r = 1$ **do**
(3) **begin synchronous round**
(4) broadcast EST1(v_i);
(5) **for each** v_j received **do** $view_i[j] \leftarrow v_j$ **end for**;
(6) **case** (equal$(\perp, view_i) \leq x$) **then** $v\_cond_i \leftarrow h(view_i)$
(7) (equal$(\perp, view_i) > x$) **then** $v\_tmf_i \leftarrow$ max(all values v_j received)
(8) **end case**;
(9) **if** $(x = t)$ **then** return(v_cond_i) **end if**
(10) **end synchronous round**;
(11) **when** $r = 2, ..., t + 1 - x$ **do**
(12) **begin synchronous round**
(13) broadcast EST2(v_cond_i, v_tmf_i);
(14) **if** $(v\_cond_i \neq \perp)$ **then** return(v_cond_i) **end if**;
(15) **if** $(v\_cond_j \neq \perp$ received during round $r)$ **then** $v\_cond_i \leftarrow v\_cond_j$ **end if**;
(16) $v\_tmf_i \leftarrow$ max(all v_tmf_j values received during r);
(17) **if** $(r = t + 1 - x)$ **then**
(18) **if** $(v\_cond_i \neq \perp)$ **then** return(v_cond_i) **else** return(v_tmf_i) **end if**
(19) **end if**
(20) **end synchronous round**.

Figure 11.9: A condition-based consensus algorithm (code for p_i)

11.3.6 Proof of the Algorithm

Theorem 49. *let C be the x-legal condition used in the algorithm described in Fig. 11.9. Let us assume the input vector $I \in C$. This algorithm implements the* consensus *agreement abstraction in the system model $CSMP_{n,t}[\emptyset]$. Moreover, no process executes more than $(t + 1 - x)$ rounds.*

Proof CC-termination. The fact that no process executes more than $(t + 1 - x)$ rounds follows directly from the synchrony assumption and the text of the algorithm (line 9 for $x = t$, and line 17-19 for $x \leq t$).

For the CC-Validity and CC-agreement properties of consensus, let us first consider the case $x = t$. As $x = t$, the non-crashed processes execute line 9. They have consequently executed the assignment $v\_cond_i \leftarrow h(view_i)$ at line 6. It then follows from the extension of the definition of $h()$ to views that, for any process p_i, we have $v\_cond_i = h(view_i) = h(I)$, which is a value that appears more than x times in I, i.e., at least once in any of the views obtained by the processes. Hence, the algorithm satisfies both the CC-validity and CC-agreement properties for $x = t$.

Let us now consider the CC-validity property for the x-legal conditions such that $x < t$. Any process p_i that terminates the first round is such that $(v\_cond_i \neq \bot) \vee (v\_tmf_i \neq \bot)$. Moreover, (for the same reasons as in the case $t = x$) if $v\_cond_i \neq \bot$, it is a value of I. Similarly, if $v\_tmf_i \neq \bot$, it is a value of I.

It follows from the text of the algorithm that, if v_cond_i is assigned at line 15, it takes the value of another non-$\bot$ v_cond_j variable, from which we conclude that any non-$\bot$ v_cond_i variable contains a value selected by $h()$ which (due to the definition of $h()$) is a value of the input vector. It follows that if a process p_i decides the value v_cond_i, it decides a value of the input vector I.

If a process p_i decides the value of v_tmf_i, it does it at line 18. In this case we have $v\_cond_i = \bot$, from which we conclude that p_i executed line 7 where v_tmf_i is assigned a proposed value. It then follows from line 16, and the fact that $\bot$ is smaller than any proposed value, that v_tmf_i always contains a proposed value. Hence, if p_i decides, it decides a proposed value.

Let us now address the CC-agreement property when $t < x$. We consider two cases.

- A process decides at line 14. Let r be the first round at which a process (say p_i) decides at line 14 of this round. Hence, p_i decides $v\_cond_i = v \neq \bot$.
  - Let us first consider the case of another process p_j that decides at line 14 of round r. Hence, p_j decides $v\_cond_i = v' \neq \bot$.
    It follows from the text of the algorithm that there are processes p_k and p_ℓ that have computed $v\_cond_k = h(view_k) = v$ and $v\_cond_\ell = h(view_\ell) = v'$ during the first round, and then these values have been propagated to p_i and p_j directly or via other processes (line 13 and line 15). (Let us observe that p_k and p_ℓ can be the same process, or can even be p_i or p_j.)
    It follows from Lemma 44, and the extension of the definition of $h()$ to views, that $h(view_x) = h(view_y)$ for any pair of processes p_x and p_y that execute line 6. Hence, we have $v = v'$ from which we conclude that no two processes that decide at line 14 during r decide differently.
  - Let us now consider the case of a process p_k that decides during a round $r' > r$. Let us observe that, at the beginning of round r, we necessarily have $v\_cond_k = \bot$ (otherwise p_k would have decided at line 14 of round r). Let us also observe that any process p_i that decides at line 14 of round r broadcast EST2$(v, -)$ before deciding. It follows that any process p_k that proceeds to round $r + 1$ is such that $v\_cond_k = v$ at the end of r (line 15). It follows from the text of the algorithm that p_k will decide $v\_cond_k = v$ during round $r + 1$ (if it does not crash). Consequently no value different from v can be decided.
- No process decides at line 14. In this case, the processes that crash terminate at line 18 of round $r = t + 1 - x$. We show that all the processes p_i that execute line 18 of round $r = t + 1 - x$ (a) have the same value in v_cond_i, and (b) have the same non-$\bot$ value in v_tmf_i, which proves the CC-agreement property for this case.

  P being the set of processes that execute line 18 of the round $r = t + 1 - x$, let us first observe that as no process $p_i \in P$ decides at line 14 during a round r, each of them has necessarily executed line 7 during the first round (otherwise we would have $v\_cond_i \neq \bot$ at the end of the first round and p_i would have decided at line 14 of the second round).

We conclude from the previous observation that, at the end of the first round, $\mathsf{equal}(\bot, view_i) > x$ and $v\_tmf_i \neq \bot$ for each process $p_i \in P$. It then follows from line 16 that these variables remain forever different from $\bot$. It also follows from $\mathsf{equal}\bot(view_i) > x$ that at least $(x+1)$ processes have crashed during the first round. This means that at most $t-(x+1)$ processes can crash from round 2 until round $t+1-x$, i.e., during $(t-x)$ rounds.

As $t-(x+1)$ processes can crash during $(t-x)$ rounds, there is necessarily a round r', $2 \leq r' \leq t+1-x$, with no crash. Moreover all the processes that execute round r' exchange their values v_cond_i and v_tmf_i (line 13). Moreover, the values v_tmf_i sent by the processes of P are not equal to $\bot$. It follows that all the processes that execute round r' have the same value in v_cond_i (this value can be $\bot$), and in v_tmf_i (this value cannot be $\bot$), which concludes the proof of the agreement property.

$\square_{Theorem\ 49}$

The next corollary follows from the proof of the previous theorem.

Corollary 5. *If at most $f \leq x$ processes crash, no process decides after the second round.*

11.4 Using a Global Clock and a Fast Failure Detector

11.4.1 Fast Perfect Failure Detectors

What is a failure detector The notion of a failure detector was introduced in Section 3.3. A failure detector is a device that provides each process with information on failures. According to the quality of this information, several classes of failure detectors can be defined.

Duration of a round To simplify the presentation, let us assume that the synchronous model is such that local computation takes no time while message transfer delays are upper bounded by duration D (a message sent at time τ is received by time $\tau + D$). The assumption that local computation takes no time is without loss of generality as processing times can be included in D. This means that the duration of a round is D time units.

The class of fast perfect failure detectors A *fast perfect failure detector* (FFD) is a distributed object that provides each process p_i with a set denoted $suspected_i$. This set contains process identities, and p_i can only read it. If $j \in suspected_i$ we say "p_i suspects p_j" or "p_j is suspected by p_i".

This object satisfies the following properties that involve a duration d, called *maximal detection time*, and is such that $d << D$ (hence the attribute *fast* of the failure detector class).

- Strong accuracy. No process p_j is suspected by another process p_i before p_j crashes.
- Detection timeliness. If a process p_j crashes at time τ, then from time $\tau + d$, every non-crashed process suspects it forever.

The first property is related to safety: no process is suspected before it crashes. The second property is related to real-time liveness. It states that a process p_i is informed of the crash of a process p_j at most d time units after the crash occurred. Let us nevertheless observe that, if a process p_j crashes at some time τ, it is possible that some processes are informed at time $\tau + d'$, while other processes are informed at time $\tau + d''$, etc., with $0 \leq d' < d'' < d$. The failure detector is *perfect* because it never makes mistakes: any crashed process is suspected, and only crashed processes are suspected. (A fast failure detector can be implemented with specialized hardware.)

11.4.2 Enriching the Synchronous Model to Benefit from a Fast Failure Detector

Instead of round numbers, the behavior of a process is described with respect to date occurrences. To this end, the synchronous system $CSMP_{n,t}[\emptyset]$ is enriched with a global clock variable denoted $CLOCK$, which a process can only read. It is assumed that $CLOCK = 0$ when the algorithm starts. Hence, the system model is $CSMP_{n,t}[CLOCK, \text{FFD}]$.

The dates are defined from the durations d (as defined by the failure detector) and D (as defined by the synchrony assumption). Hence, they are meaningful both from the application point of view (D) and the failure detector point of view (d). A particular algorithm defines which are the dates that are relevant for it.

11.4.3 A Simple Consensus Algorithm Based on a Fast Failure Detector

Considering the model $CSMP_{n,t}[CLOCK, \text{FFD}]$, the algorithm described in Fig. 11.10 allows the processes to decide at time $t \times d + D$. This is better than its counterpart in a pure synchronous system which requires $(t+1)$ rounds, i.e., $(t+1)D$ times units.

Relevant dates The algorithm considers two types of rounds, rounds of duration D time units as defined by the synchronous system, and rounds (called FFD-rounds) of duration d (maximal detection time) related to the underlying failure detector. According to these rounds, the dates that are relevant for a process p_i are $(i-1)d$ for sending a message (line 2) and $t \times d + D$ for deciding (line 5).

Description of the algorithm The principle the algorithm relies on is the following. Each FFD-round is coordinated by a process that is the only process allowed to send a message during this FFD-round (lines 2-3). Process p_1 is the coordinator of the first FFD-round, process p_2 the coordinator of the second FFD-round, etc. More precisely, at the beginning of the FFD-round $(i-1)d$, process p_i is required to broadcast the pair (est_i, i) (where est_i is its current estimate of the decision value) if, and only if, it suspects all the processes that were assumed to broadcast during the previous FFD-rounds (i.e., if it suspects the processes p_1 to p_{i-1}). Let us observe that, if p_1 does not crash, its broadcast predicate is trivially satisfied when the algorithm starts (i.e., when $CLOCK = 0$).

If any, the message broadcast by a process p_i is sent at time $(i-1)d$ and received by time $(i-1)d + D$. If p_i crashes during the broadcast, an arbitrary subset of processes receive its message, and if p_i crashes at time τ, a process p_j starts suspecting p_i forever at any time between τ and $\tau + d$. When a process p_i receives a message, it stores the pair contained in the message into a set denoted $view_i$ (line 4). If a message is received by a process p_i when a relevant date occurs for it (i.e., when $CLOCK = (i-1)d$ or $CLOCK = t \times d + D$), this process first processes the message received (which by assumption takes no time), and only then executes the statement associated with the corresponding date.

Finally, at time $t \times d + D$ (line 5), any alive process p_i decides and stops. The value it decides is the value it has received that has been sent by the process with the highest identity.

Remark As at most t processes crash, the processes p_{t+2}, ..., p_n can never be round coordinators, and consequently their values can never be decided (except when one of their values is also proposed by a process p_x with $1 \leq x \leq t+1$). The algorithm is consequently unfair in the sense given in Section 10.1.2.

Theorem 50. *The algorithm described in Fig. 11.10 implements the* consensus *agreement abstraction in the system model* $CSMP_{n,t}[CLOCK, \text{FFD}]$. *Moreover, the decision is obtained in* $t \times d + D$ *time units.*

```
operation propose(v_i) is
  init est_i ← v_i; view_i ← ∅.

  when CLOCK = (i − 1)d do
    if ({1, 2, ..., i − 1} ⊆ suspected_i) then broadcast EST(est_i, i) end if.

  when EST(est, j) is received do view_i ← view_i ∪ {⟨est, j⟩}.

  when CLOCK = t × d + D do
    let ⟨v, k⟩ be the pair in view_i with the greatest process identity;
    return(v).
```

Figure 11.10: Synchronous consensus with a fast failure detector (code for p_i)

Proof The CC-termination property follows from the synchrony assumptions of the synchronous system and the underlying failure detector: when the clock is equal to $t \times d + D$, all alive processes decide. Moreover, when a process p_i decides, $view_i$ is not empty (because there is at least one correct process among the $(t+1)$ coordinators), and contains only proposed values. Hence, the CC-validity property is also met.

To prove the CC-agreement property we first introduce a definition and then prove a claim from which CC-agreement is derived.
Definition. An FFD-round k is *eligible* if, at time $(k-1)d$, the processes p_1, ..., p_{k-1} have crashed and p_k either crashed or suspects them.

Let us observe that, if the FFD-round $(t+1)$ is eligible, then process p_{t+1} must be alive at time $td + D$. This is because at most t processes can crash, and, as the FFD-round $(t+1)$ is eligible, the processes p_1 to p_t have crashed. Let us also observe that no FFD-round $k > t+1$ can be eligible. Finally, let us notice that, due to the definition of eligibility, a process p_i can broadcast a message in the FFD-round i only if this FFD-round is eligible.

Claim. For $1 \leq k \leq t+1$, if the FFD-round k is eligible, then either p_k broadcasts $\text{EST}(est_i, v)$ or the round $(k+1)$ is eligible.
Proof of the claim. If the FFD-round k is eligible and p_k does not broadcast $\text{EST}(est_i, v)$, then p_k crashes by time $(k-1)d$. In this case, due to the detection timeliness of the failure detector, it will be suspected by all alive processes by time $(k-1)d + d = k \times d$, and then the FFD-round $(k+1)$ is eligible. End of the proof of the claim.

Let us now prove the CC-agreement property. Let r be the largest eligible FFD-round. It follows from the previous discussion that $r \leq t+1$. It then follows from the claim that process p_r sends $\text{EST}(est_r, v)$ to all other processes without crashing (otherwise r would not be the largest eligible FFD-round). Moreover, no process with a larger identity ever broadcasts a message (this is because for p_j to broadcast a message, the FFD-round j has to be eligible, and r is the largest eligible round). It follows that all processes that decide at time $t \times d + D$, decide the value est_r they have received, which concludes the proof of the theorem. $\square_{Theorem\ 50}$

11.4.4 An Early Deciding and Stopping Algorithm

Decide in $f \times d + D$ time units Let us remember that f, $0 \leq f \leq t$, denotes the actual number of process crashes in an execution. This section presents a consensus algorithm suited to the model $CSMP_{n,t}[CLOCK, \text{FFD}]$, in which any process (that does not crash) decides by $D + fd$ time units. This is better than $\min(f+2, t+1)D$ time units which is the bound attained by the early deciding algorithm presented in Section 11.1. To simplify the presentation, it is assumed that D is an integral multiple of d.

Local variables at process p_i Each process p_i manages two local variables:

- est_i is p_i's estimate of the decision value. Its initial value is v_i, the value proposed by p_i,.
- max_id_i contains a process identity. Its initial value is 0 (any value smaller than a process identity).

Relevant dates The algorithm is described in Fig. 11.12. It is an extension of the previous fast failure detector-based algorithm. It has consequently the same coordinator-based sequential structure. More precisely, it also considers periods of length d, each coordinated by a process: process p_i is the only process that can send a message at the beginning of the time period defined by the clock interval $[(i-1)d..i \times d)$ (lines 2-3 are the same as in Fig. 11.10). Hence, as before, the first period is coordinated by p_1, the second by p_2, etc. Therefore, the dates that are relevant for this algorithm are: D, $d+D$, $2d+D$, ..., $t \times d + D$ for all processes (line 6), plus the date $(i-1)d$ for each process p_i (line 2). These dates are represented on Fig. 11.11.

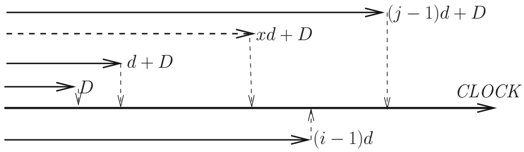


Figure 11.11: Relevant dates for process p_i

Early deciding fast failure detector-based algorithm As already mentioned, the statements executed by p_i when $CLOCK = (i-1)d$ (lines 2-3) are the same as in Fig. 11.10: if p_i suspects all the processes with a smaller identity, it sends the pair (est_i, i) to all processes.

The statements executed by a process p_i when it receives a message or when $CLOCK = (j-1)d+D$ are different from the ones in the previous algorithm. When process p_i receives a pair (est, j) it updates its own estimate est_i (line 5) only if the identity j of the sender process is larger than max_id_i (which has been initialized to a value smaller than any process identity). Hence, except for its initial value, the successive values of est_i come from processes with increasing identities.

Finally, at every date $(j-1)d+D$, $1 \leq j \leq t+1$ (line 6), p_i checks a predicate to see if it can decide. This predicate is on the current output of the failure detector. More precisely, p_i decides if it does not suspect the process p_j currently defined from the value of the clock. If the predicate is false, p_i received the message (if any) sent by p_j. (This is because the difference between its sending time and the current time is D. Moreover, if p_j has not sent a message, it is because it did not suspect at least one of its predecessors p_1 to p_{j-1}.) Hence, if $j \notin suspected_i$, p_i decides the current value of est_i and consequently executes return(est_i) (line 7).

It is easy to see that the processes decide by D time units when the process p_1 does not crash (in that case they decide the value v_1 proposed by p_1). If p_1 crashes while p_2 does not, they decide by time $d+D$. According to the failure pattern, the decided value is then the value v_1 proposed by p_1 or the value v_2 proposed by p_2 (it is v_1 if p_2 has received v_1 by d time units), etc.

Theorem 51. *The algorithm described in Fig. 11.12 implements the* consensus *agreement abstraction in the system model $CSMP_{n,t}[CLOCK, \text{FFD}]$. Moreover, the decision is obtained in at most $f \times d + D$ time units, where f is the actual number of process crashes.*

```
operation propose(v_i) is
init est_i ← v_i; max_id_i ← 0.

when CLOCK = (i − 1)d do
   if ({1, 2, ..., i − 1} ⊆ suspected_i) then broadcast EST(est_i, i) end if.

when EST(est, j) is received do
   if (j > max_id_i) then est_i ← est; max_i ← j end if.

when CLOCK = (j − 1)d + D for every 1 ≤ j ≤ t + 1 do
   if (j ∉ suspected_i) then return(est_i) end if.
```

Figure 11.12: Early deciding synchronous consensus with a fast failure detector (code for p_i)

Proof Let us first observe that no process p_i decides after $d \times f + D$ times units. Indeed, as f processes crash and $f \leq t$, there is at least one process p_j such that $1 \leq j \leq t+1$ and the predicate $j \notin suspected_i$ is consequently satisfied at the latest when when $CLOCK = (j-1)d + D$. The CC-termination property follows from this observation. Moreover, the CC-validity property is trivial (for any p_i, est_i is initialized to v_i, and then possibly updated only with another estimate value).

The proof of the CC-agreement property is based on the following definition.
Definition. An FFD-round k is *active* if, at time $(k-1)d$, p_k is not crashed and suspects the processes p_1, ..., p_{k-1}. Let us observe that an active FFD-round is eligible, while an eligible FFD-round is not necessarily active.

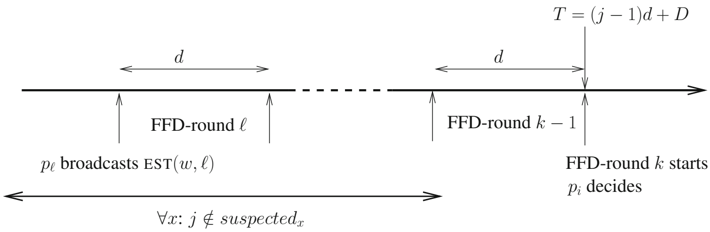


Figure 11.13: The pattern used in the proof of the CC-agreement property

The timing pattern used in the proof is described in Fig. 11.13.

- Let us consider the first process (say p_i) that decides. Let v be the value it decides. Process p_i has decided v at some time $T = (j-1)d + D$ for some j. It follows from the failure detector-based decision predicate that, at time T, process p_i was not suspecting p_j. It follows from the detection timeliness property of the failure detector that no process suspected p_j at least up to time $T - d$ (Observation O1).
- Due to the simplifying assumption that D is an integral multiple of d, it follows that there is an FFD-round k that starts at time T. Moreover, (due to O1) no process suspected p_j at the beginning of every FFD-round $x < k$ (Observation O2).
- Due to the definition of "active FFD-round" and O2, it follows that none of the rounds from $(j+1)$ until $(k-1)$ are active (Observation O3).
- On the other hand, as p_j is alive at time $T-d$ (see O1), and $T-d = (j-1)d+D-d > (j-1)d$, process p_j is alive at time $(j-1)d$ (Observation O4).
- It follows that there is at least one active FFD-round among the FFD-rounds 1 to j. The only way for none of these FFD-rounds be active is that for any x in $\{1, \ldots, j\}$ process p_x crashes at

time $(x-1)d$, and we know from O4 that this is false at least for p_j. Hence, there is a largest active FFD-round – say ℓ – in the FFD-rounds from 1 to j (Observation O5).

- It follows from the text of the algorithm and the definition of an active FFD-round that p_ℓ (which exists due to O5) broadcast $\text{EST}(w,\ell)$ at the beginning of the FFD-round ℓ, and this message is received by all the processes by time $(\ell-1)d+D<T$ (Observation O6).
- It follows from the choice of ℓ and O3 that there are no active FFD-rounds among the FFD-rounds from $(\ell+1)$ to $(k-1)$. Consequently, none of the processes from $p_{\ell+1}$ to p_{k-1} sends messages (Observation O7).
- It follows from O6 that, at time T, all processes have received $\text{EST}(w,\ell)$ and changed their est_i variable to w. Moreover, due to O7, est_i is not overwritten. Hence, at time T, no estimate value of an alive process is different from w. It follows that, whatever the messages sent after T, all estimates remain equal to w. Hence, $v=w$, and no decided value can be different from w.

$\square_{Theorem\ 51}$

On the failure detector behavior Let us observe that when a process p_i decides, it stops its execution as far as consensus is concerned but it continues executing the program it is involved in. If process p_i crashes later (i.e., outside the consensus algorithm), the failure detector detects its crash, and this detection does not alter the correction of the consensus algorithm. Whereas, if p_i terminates, the failure detector must not consider its normal termination as a crash (such a false detection could make the consensus algorithm incorrect). The failure detector detects crash failures and only crash failures. A normal termination is not a failure.

11.5 Summary

This chapter was devoted to efficient consensus algorithms, where efficiency concerns the number of rounds executed by an algorithm. Two algorithms ensuring that no process executes more than $\mathsf{min}(f+2,t+1)$ have been presented. One is based on the counting of crashed processes, the other one is based on a differential predicate, which provides a finer view of the execution and can be exploited to favor early decision.

Then, the chapter presented an unbeatable predicate, and the associated consensus algorithm CGM. Unbeatability means that, if there is an early deciding algorithm A based on a different decision predicate that, in some execution, improves the decision round with respect to CGM, there is at least one execution of A in which a process strictly decides later than in CGM.

Finally, the chapter has presented the condition-based approach which allow us to bypass the lower bound $\mathsf{min}(f+2,t+1)$ when the set of possible input vectors satisfies some predefined pattern, and the enrichment of a synchronous system with a fast failure detector, which allows us to expedite decision.

11.6 Bibliographic Notes

- Early deciding agreement was first investigated by D. Dolev, R. Reischuk, and H.R. Strong in [135].
- The predicate for early interactive consistency used in Section 11.1.2 and the corresponding early deciding and stopping algorithm are from [362].
- The early decision lower bound on the number of rounds for consensus is $f+2$ when $f<t-1$ and $f+1$ when $f\geq t-1$ (e.g., [106, 246, 411]). By an abuse of notation, this lower bound is usually denoted $\mathsf{min}(f+2,t+1)$ (the special case is when $f=t-1$).

- The notion of unbeatability is from [209] (where it is called optimality). Knowledge theory is developed in [152]. The unbeatable binary consensus predicate and the associated algorithm are due to A. Castañeda, Y. Gonczarowski, and Y. Moses [92]. The presentation adopted in Section 11.2 is from [99].

  It is shown in [302] that there is no "all cases" optimal predicate for early deciding consensus.

  A similar unbeatability result presented in [141] holds for the non-blocking atomic commit problem [192, 193]. This problem will be the topic addressed in Chap. 13)
- The condition-based approach was introduced by A. Mostéfaoui, S. Rajsbaum and M. Raynal in [313], where it is shown that x-legality is a necessary and sufficient property to solve consensus in an asynchronous system prone to up to x process crashes.
- The condition-based approach was extended to synchronous system by the same authors in [314] where is presented the hierarchy of conditions for synchronous systems.

  This paper also presents an early deciding condition-based consensus algorithm that does not require that the input vector always belongs to the x-legal condition C it is instantiated with. This algorithm directs the processes to decide in at most $\mathsf{min}(f+2, t+1-x)$ rounds in all the executions whose input vector I belongs to C, and in at most $\mathsf{min}(f+2, t+1)$ rounds if $I \notin C$.
- The condition-based approach was extended to the interactive consistency problem in [315].
- The relation between agreement problems and error-correcting codes is due to R. Friedman, A. Mostéfaoui, S. Rajsbaum, and M. Raynal [167]. More developments on the condition-based approach to solve agreement problems can be found in [238, 239, 316, 318, 420].
- Failure detectors were introduced by T. Chandra, V. Hadzilacos, and S. Toueg in [101, 102], where they are used to circumvent the impossibility to solve consensus in asynchronous systems prone to process crash failures [162]. Introductory surveys to failure detectors can be found in [195, 365].
- Fast failure detectors were introduced by M. Aguilera, G. Le Lann, and S. Toueg in [19] along with the algorithms presented in this chapter.

11.7 Exercises and Problems

1. Prove the early deciding consensus algorithm described in Fig. 11.3.
2. Let us consider the unbeatable binary consensus algorithm described in Fig. 11.7.
   - Let lg_i^r be the value of the graph lg_i at the end of round r. Prove (by induction) that lg_i^r captures the causal past of p_i at the end of round r (round invariant of the algorithm in Fig. 11.7).
   - With the help of the previous round invariant, prove the CC-agreement property of the unbeatable algorithm described in Fig. 11.7.
3. Prove that the condition C^x_{first} defined in Section 11.3.2 is x-legal. Show it is not maximal.

   Solution in [313].