Looking for the Weakest Failure Detector for k-Set Agreement in Message-passing Systems: Is Π_k the End of the Road?

François Bonnet, Michel Raynal

Publications Internes de l'IRISA
ISSN : en cours
PI 1929 – mai 2009

Looking for the Weakest Failure Detector for k-Set Agreement in Message-passing Systems: Is Π_k the End of the Road?

François Bonnet[*] , Michel Raynal[**]
francois.bonnet@irisa.fr, raynal@irisa.fr

Abstract: In the k-set agreement problem, each process (in a set of n processes) proposes a value and has to decide a proposed value in such a way that at most k different values are decided. While this problem can easily be solved in asynchronous systems prone to t process crashes when $k > t$, it cannot be solved when $k \leq t$. Since several years, the failure detector-based approach has been investigated to circumvent this impossibility. While the weakest failure detector class to solve the k-set agreement problem in read/write shared-memory systems has recently been discovered (PODC 2009), the situation is different in message-passing systems where the weakest failure detector classes are known only for the extreme cases $k = 1$ (consensus) and $k = n - 1$ (set agreement). This paper introduces a candidate for the general case. It presents a new failure detector class, denoted Π_k, and shows $\Pi_1 = \Sigma \times \Omega$ (the weakest class for $k = 1$), and $\Pi_{n-1} = \mathcal{L}$ (the weakest class for $k = n - 1$). Then, the paper investigates the structure of Π_k and shows it is the combination of two failures detector classes denoted Σ_k and Ω_k (that generalize the previous "quorums" and "eventual leaders" failure detectors classes). Finally, the paper proves that Σ_k is a necessary requirement (as far as information on failure is concerned) to solve the k-set agreement problem in message-passing systems. The paper presents also a Π_{n-1}-based algorithm that solves the $(n - 1)$-set agreement problem. This algorithm provides us with a new algorithmic insight on the way the $(n - 1)$-set agreeement problem can be solved in asynchronous message-passing systems (insight from the point of view of the non-partitioning constraint defined by Σ_{n-1}).

Key-words: Asynchronous systems, Eventual leaders, Failure detectors, Message passing system, Quorums, Reduction, k-Set agreement, Wait-freedom.

A la recherche du plus faible détecteur de fautes pour l'accord k-ensembliste dans les systèmes à passage de messages

Résumé : *Ce rapport étudie le problème du plus faible détecteur de fautes pour l'accord k-ensembliste dans les systèmes asynchrones à passage de messages sujets aux défaillances des processeurs.*

Mots clés : *Accord ensembliste, Asynchronisme, Défaillances des processus, Détecteur de fautes.*

---

[*] Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'universit Rennes 1 et l'INSA de Rennes
[**] Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'universit Rennes 1 et l'INSA de Rennes

1 Introduction

The k-set agreement problem This problem is a coordination problem (also called *decision* task). It involves n processes and is defined as follows [5]. Each process proposes a value and every non-faulty process has to decide a value (termination), in a such a way that any decided value is a proposed value (validity) and no more than k different values are decided (agreement). The problem parameter k defines the coordination degree; $k = 1$ corresponds to its most constrained instance (consensus problem) while $k = n - 1$ corresponds to its weakest non-trivial instance (set consensus problem).

Considering the process crash failure model, let t be the maximal number of processes that may crash in a run ($1 \leq t < n$). When $t < k$, the k-set agreement can always be solved, be the system synchronous or asynchronous. When $t \geq k$, the situation is different. While the problem can always be solved in synchronous systems, [6] (see [25] for a survey), it has no solution in asynchronous systems [2, 17, 27].

The failure detector-based approach A failure detector is a distributed oracle that gives alive processes hints on process failures [3]. Failure detectors have been investigated to solve k-set agreement problem since 2000 [21][1]. Lower bounds to solve the *k*-set agreement in asynchronous message-passing systems enriched with limited accuracy failure detectors have been conjectured in [21] and proved in [16]. The question of the weakest failure detector class for the k-set agreement problem ($k > 1$) has been stated first in [24].

The case $k = 1$ and the case $k = n - 1$ When $k = 1$, as already indicated k-set agreement boils down to consensus, and it is know that the failure detector class Ω is the weakest to solve consensus in asynchronous message-passing systems where $t < n/2$ [4]. Ω ensures that there is an unknown but finite time after which all the processes have the same non-faulty leader (before that time, there is an anarchy period during which each process can have an arbitrarily changing leader). This lower bound result is generalized in [10] where it is shown that $\Sigma \times \Omega$ is the weakest failure detector class to solve consensus when $t < n$. This means that Σ is the minimal additional power (as far as information on failures is concerned) required to overcome the barrier $t < n/2$ and attain $t \leq n-1$. Actually the power provided by Σ is the minimal one required to implement a shared register in a message-passing system [9, 10]. Σ provides each process with a quorum (set of process identities) such that the values of any two quorums (each taken at any time) intersect, and there is a finite time after which any quorum includes only correct processes [9]. Fundamentally, Σ prevents partitioning. A failure detector of the class $\Sigma \times \Omega$ outputs a pair of values, one for Σ and one for Ω.

The weakest failure detector classes for the $(n - 1)$-set agreement have been established in 2008, and surprisingly they are not the same in the shared memory model and the message-passing model. More precisely, the weakest class for solving the $(n - 1)$-set agreement problem in the asynchronous read/write shared memory model is Anti-Ω (denoted here $\overline{\Omega}_{n-1}$) [28]. Such a failure detector provides each process with a set of $(n - 1)$ "leaders" that can change with time but these sets are such that, after some unknown but finite time, they all contain the same non-faulty process[2].

Differently, the weakest class for solving $(n - 1)$-set agreement in the asynchronous message-passing model, is the *Loneliness* failure detector class (denoted $\mathcal{L}$) [11]. Such a failure detector provides each process p with a boolean (that p can only read) such that the boolean of at least one process remains always false and, if all but one process crash, the boolean of that process becomes and remains true forever.

The general case for read/write shared memory The failure detector class $\overline{\Omega}_k$ has first been presented at the PODC'07 rump session [26] where it has been conjectured to be the weakest failure detector class for solving the k-set agreement problem in read/write shared memory systems. This conjecture has been very recently (PODC 2009) proved by three independent groups [12, 13, 14] (using apparently very different techniques). A failure detector of the class $\overline{\Omega}_k$ provides each process with a (possibly always changing) set of k processes such that after some unknown but finite time all the sets that are output have in common the same non-faulty process.

The optimality of $\overline{\Omega}_k$ to solve k-set agreement in shared memory systems seems to be related to the fact that this problem is equivalent to the k-simultaneous consensus problem [1], in which each process executes k independent consensus instances (to which it proposes the same input value), and is required to terminate in one of them. As shown in [28], this problem has been instrumental in determining the weakest failure detector for wait-free solving the $(n - 1)$-set agreement problem in asynchronous shared memory systems.

Content of the paper This paper proposes and investigates a new failure detector class for solving the k-set agreement problem in asynchronous message-passing systems. Its main contributions are the following.

- A new family of failure detector classes, denoted $\{\Pi_k\}_{1 \leq k < n}$, is introduced. Its first interest lies in the fact that (1) $\Pi_1 \simeq \Sigma \times \Omega$ (i.e., it allows expressing the weakest failure detector class for consensus with a one-dimensional output, namely a set of process identities), and (2) $\Pi_{n-1} = \mathcal{L}$, from which it results that Π_k is optimal for the extreme values of k when one wants to solve the k-set agreement problem in message-passing systems. Expressing the power of both $\Sigma \times \Omega$ and $\mathcal{L}$ with a single formalism was not a priori evident.

[1] Similarly to consensus, the randomized approach also has been investigated to solve the k-set agreement problem [22].
[2] Anti-Ω is defined in a different but equivalent way in [28].

- It is shown that the class Π_k is actually equivalent to the class $\Sigma_k \times \Omega_k$ where Σ_k is an appropriate generalization of Σ.[3] We have $\Sigma_1 \equiv \Sigma$, and very interestingly $\Pi_{n-1} \simeq \Sigma_{n-1} \simeq \mathcal{L}$ which sheds a new light on the weakest failure detector class for the $(n-1)$-set agreement problem.
- It is proved that for any k, Σ_k is a necessary requirement (as far as information on failures is concerned) to solve the k-set agreement problem in message-passing systems. It is worth noticing that the proof of this necessity requirement does rely neither on an heavy machinery, nor on a reduction to a previous impossibility result. It is purely constructive and particularly simple.

The paper additionally presents a message-passing $(n-1)$-set agreement algorithm directly based on Π_{n-1} (i.e., Σ_{n-1}). As already indicated, this provides us with a new algorithmic insight on the way the $(n-1)$-set agreement can be optimally solved.

Last but not least, an output of this paper is the following intriguing question. As already indicated, the k-set agreement problem and the k-simultaneous consensus problem are equivalent in read/write shared memory systems [1], which means that k-set agreement can be solved by executing k independent consensus instances. From a "minimal information on failures" point of view, each such instance relies on the shared memory (i.e., on Σ) to ensure agreement, and on an instance of Ω to ensure termination. For the k-set agreement we only need that one instance does terminate. This is what is captured by $\overline{\Omega}_k$ (that eventually provides the processes with sets of k leaders that can arbitrarily change but contain forever the same correct process).

So, the question is: Which is the relation between the k-set agreement problem and the k-simultaneous consensus problem in message-passing systems? Understanding this link and its nature would give us a better understanding of the fundamental difference between shared memory communication and message-passing communication. The intertwining between sharing and agreeing seems to be subtle [8].

Roadmap This paper is made up of 8 sections. Section 2 describes the computation model and Section 3 defines the failure detector class Π_k. Then, Section 4 shows that the classes $\{\Pi_k\}$ and $\Sigma_k \times \Omega_k$ are equivalent, and Section 5 shows that Π_{n-1} and $\mathcal{L}$ are equivalent. Section 6 presents a Π_{n-1}-based $(n-1)$-set agreement algorithm. Section 7 proves that Σ_k is a necessary requirement for failure detector-based k-set agreement in message-passing systems. Finally, Section 8 concludes the paper.

2 System model and k-set agreement

2.1 System model

Process model The system consists of a set of $n > 2$ asynchronous processes denoted $P = \{p_1, \ldots, p_n\}$. Each process executes a sequence of atomic steps (internal action, sending of a message, or reception of a message). A process executes its code until it possibly crashes. After it has crashed a process executes no more step. A process that crashes during a run is *faulty* in that run, otherwise it is *correct*. Given a run, $\mathcal{C}$ denotes the set of processes that are correct in that run. Up to $(n-1)$ processes can crash in a run. This is called the *wait-free environment*.

Communication model The processes communicate by sending and receiving messages through channels. Every pair of processes is connected by a bidirectional channel. The channels are failure-free (there is no creation, alteration, duplication or loss of messages) and asynchronous (albeit the time taken by a message to travel from its sender to its destination process is finite, there is no bound on transfer delays). The notation "broadcast $\text{MSG\_TYPE}(m)$" is used to send a message m (the type of which is MSG_TYPE) to all the processes. It is a (non-atomic) shortcut for "**for each** $j \in \{1, \ldots, n\}$ **do** send $\text{MSG\_TYPE}(m)$ to p_j **end for**".

Notation The previous asynchronous message-passing model is denoted $\mathcal{AS}_n[\emptyset]$. When enriched with any failure detector of a given class X, it will be denoted $\mathcal{AS}_n[X]$.

2.2 The k-set agreement problem

As already indicated, the k-set agreement problem has been introduced by S. Chaudhuri [5]. It generalizes the consensus problem (that corresponds to $k = 1$). It is defined as follows. Each process proposes a value and has to decide a value in such a way that the following properties are satisfied:

- Termination. Every correct process decides a value.
- Validity. A decided value is a proposed value.
- Agreement. At most k different values are decided.

[3]Interestingly, a failure detector class weaker than $\Sigma \times \Omega_k$ is proposed in [7] to solve k-set agreement in message-passing systems. It is easy to show that $\Sigma \times \Omega_{n-1}$ is stronger than $\mathcal{L}$.

3 Failure detector classes definition

If xx_i is the local variable that contains the output of the failure detector at process p_i, xx_i^τ denotes its value at at time τ.

3.1 The eventual leaders families (the Omega families)

Each process p_i is endowed with a local variable $leaders_i$ that satisfies the following properties.

The eventual leaders family Ω_k ($1 \leq k \leq n-1$) This family has been introduced by Neiger [23]. The local variables $leaders_i$ satisfy the following properties.

- Validity. $\forall i : \forall \tau : leaders_i^\tau$ is a set of k process identities.
- Eventual leadership. $\exists \tau : \exists LD = \{\ell_1, \ldots, \ell_k\} : (LD \cap \mathcal{C} \neq \emptyset) \wedge (\forall \tau' \geq \tau : \forall i : leaders_i^{\tau'} = LD)$.

Let us notice that τ is finite but unknown. Before τ, there is an anarchy period during which the local sets $leaders_i$ can contain unrelated values. After τ, these sets are equal to the same set LD that contains at least one correct process.

$\Omega = \Omega_1$ is the weakest failure detector class to solve consensus [4] in message-passing systems with a majority of correct processes, and in shared memory systems [15, 18]. An Ω_k-based algorithm that solves the k-set agreement in message-passing systems where $t < n/2$ is described in [20]. This algorithm can easily be modified to replace the $t < n/2$ assumption by a failure detector of the class Σ_1 (as defined below [9]).

The eventual leaders family $\overline{\Omega}_k$ ($1 \leq k \leq n-1$) The class $\overline{\Omega}_{n-1}$ (called anti-Omega) has been introduced in [28] where it has been shown to be weakest failure detector class to solve $(n-1)$-set agreement in shared memory systems. It has been generalized in [26] (as cited in [28]). The local variables $leaders_i$ satisfy the following properties.

- Validity. $\forall i : \forall \tau : leaders_i^\tau$ is a set of k process identities.
- Weak Eventual leadership. $\exists \tau : \exists \ell \in \mathcal{C} : \forall \tau' \geq \tau : \forall i : \ell \in leaders_i^{\tau'}$.

$\overline{\Omega}_1$ is the same as Ω_1. For $k > 1$, $\overline{\Omega}_k$ is weaker than Ω_k: it requires only that after some (finite but unknown) time the sets $leaders_i$ contain the same correct process. Very recently, it has been shown that $\overline{\Omega}_k$ is the weakest failure detector class to solve k-set agreement in shared memory systems [12, 13, 14]. As noticed in the Introduction, this family of failure detectors is related to the k-set consensus problem [1].

3.2 The quorum family Σ_k ($1 \leq k \leq n-1$)

Each process p_i is endowed with a local variable qr_i that satisfies the following properties.

- Intersection. Let $\{id_1, \ldots, id_{k+1}\}$ denote a subset of $k+1$ process identities, and $\tau_1, \ldots, \tau_{k+1}$ be any multiset of $k+1$ arbitrary time instants. $\forall \{id_1, \ldots, id_{k+1}\} : \forall \{\tau_1, \ldots, \tau_{k+1}\} : \exists i, j : 1 \leq i \neq j \leq k+1 : (qr_{id_i}^{\tau_i} \cap qr_{id_j}^{\tau_j} \neq \emptyset)$.
- Liveness. $\exists \tau : \forall \tau' \geq \tau : \forall i \in \mathcal{C} : qr_i^{\tau'} \subseteq \mathcal{C}$.

After a process p_i has crashed (if it ever does), we have (by definition) $qr_i = \{1, \ldots, n\}$ forever.

Σ_k is a generalization of the quorum failure detector class Σ introduced in [10] (that does correspond to Σ_1), where it is shown to be the weakest failure detector class to implement an atomic register in a message-passing system whatever the number of process failures ("wait-free" environment). It is interesting to notice that the intersection property of Σ_k is the same as the one used to define k-coteries [19].

3.3 The agreement quorum family Π_k ($1 \leq k \leq n-1$)

Each process p_i is endowed with a local variable qr_i that satisfies the Intersection and Liveness properties of the quorum family Σ_k plus the following property:

- Eventual leadership. $\exists \tau : \exists LD = \{\ell_1, \ldots, \ell_k\} : \forall \tau' \geq \tau : \forall i : qr_i^{\tau'} \cap LD \neq \emptyset$.

After a process p_i has crashed (if it ever does), we have (by definition) $qr_i = \{1, \ldots, n\}$ forever. Moreover, let us observe that the Eventual leadership property of Π_k is weaker than the Eventual leadership property of Ω_k or $\overline{\Omega}_k$: it is not required that, after τ, qr_i must always contain the same correct process.

It follows from the Intersection property that a quorum can never be empty. Moreover, it follows from the Liveness property that the set $LD = \{\ell_1, \ldots, \ell_k\}$ defined in the Eventual leadership property is such that $LD \cap \mathcal{C} \neq \emptyset$ (which means that this set contains at least one correct process). Let us also observe that the intersection requirement in the Eventual leadership property is similar to but weaker than the intersection property used in the definition of a k-arbiter [19].

3.4 Relations between failure detector classes

Definition 1 *The failure detector class A is* stronger *than the failure detector class B (denoted $A \succeq B$ or $B \preceq A$) if it is possible to build a failure detector of the class B in $\mathcal{AS}_n[A]$.*

It follows from their definitions that (1) for any k: $\Omega_k \succeq \overline{\Omega}_k$, and (2) FD standing for any of Σ, Ω, $\overline{\Omega}$, and Π: $FD_1 \succeq \cdots FD_k \succeq FD_{k+1} \cdots \succeq FD_{n-1}$.

Definition 2 *The class A is* strictly stronger *than the class B (denoted $A \succ B$) if $A \succeq B$ and $\neg(B \succeq A)$.*

Definition 3 *The classes A and B are* equivalent *(denoted $A \simeq B$) if $A \succeq B$ and $B \succeq A$.*

4 Π_k vs $\Sigma_k \times \Omega_k$ ($1 \le k \le n-1$)

4.1 From $\Sigma_k \times \Omega_k$ to Π_k

An algorithm that builds a failure detector of the class Π_k from a failure detector of the class $\Sigma_k \times \Omega_k$ is described in Figure 1.

Init: $queue_i \leftarrow < 1, \ldots, n >$.

Task $T1$: **repeat periodically** broadcast ALIVE(i) **end repeat**.

Task $T2$: **when** ALIVE (j) **is received**: suppress j from $queue_i$; enqueue j at the head of $queue_i$.
when p_i reads qr_i: **let** ℓ **be** the first id of $queue_i$ that belongs to the output of Ω_k;
return (output of $\Sigma_k \cup \{\ell\}$).

Figure 1: From $\Sigma_k \times \Omega_k$ to Π_k (code for p_i)

Theorem 1 *The algorithm described in Figure 1 is a wait-free construction of a failure detector of the class Π_k in $\mathcal{AS}_n[\Sigma_k \times \Omega_k]$.*

Proof The Intersection property of Π_k follows directly from the corresponding property of Σ_k and the fact that qr_i includes the current output of Σ_k.

For Liveness property of Π_k let us recall that after some finite time τ, Ω_k outputs forever the same set $\{\ell_1, \ldots, \ell_k\}$ of k process identities and this set contains at least one correct process. Let us consider any time instant after τ, and a correct process p_i. Due to the ALIVE (j) messages periodically sent by the correct processes, it follows that the ids of correct processes move at the head of $queue_i$ (see task $T2$). It follows that the process p_ℓ that is currently selected by the task $T2$ is always a correct process locally output by Ω_k. This, combined with the fact that there is a time after which Σ_k always outputs correct processes, proves the Liveness property of Π_k.

The Eventual leadership property of Π_k follows directly from the fact that, after some finite time, Ω_k always outputs the same set $\{\ell_1, \ldots, \ell_k\}$ of k process identities, and the fact that one of these identities appears in the definition of the current value of qr_i.
$\Box_{Theorem\ 1}$

4.2 From Π_k to Σ_k and Ω_k

It is trivial to build Σ_k in $\mathcal{AS}_n[\Pi_k]$: the output of Σ_k is the output of Π_k. The rest of this section focuses on the construction of Ω_k in $\mathcal{AS}_n[\Pi_k]$.

4.2.1 Description of the algorithm

Principle of the algorithm Each process p_i manages a local variable $quorum\_set_i$ that contains a set of quorums. (Its initial value is the current value of qr_i, the local output supplied by Π_k). The principle of the algorithm is to maintain invariant the following property where $\ell_1, \ldots, \ell_k$ are different process identities:

$$(\exists\{\ell_1, \ldots, \ell_k\} : \forall qr \in quorum\_set_i : qr \cap \{\ell_1, \ldots, \ell_k\} \neq \emptyset),$$

and "extract" Ω_k from it.

As we are about to see, this property guarantees that, if the process p_i was alone, it could consider $\{\ell_1, \ldots, \ell_k\}$ as its local output of Ω_k. So, in addition of maintaining the previous property invariant, the processes additional use a reset mechanism and a gossip mechanism in order to ensure that all the local outputs ($\{\ell_1, \ldots, \ell_k\}$) eventually satisfy the leadership property of Ω_k.

Description of the algorithm The algorithm is described in Figure 2 in which each **when** statement is assumed to be executed atomically. Each process p_i executes a sequence of phases, locally identified by ph_nb_i. The behavior of p_i is as follows.

- Initially, p_i broadcasts $\text{NEW}(quorum\_set_i, ph\_nb_i)$ to inform the other processes of its value qr_i locally supplied by Π_k. It does the same broadcast each time the value of $quorum\_set_i$ changes (line 15 whose execution is entailed by the invocation of $\mathsf{pres\_inv\&gossip}()$ at lines 02 or 07).
- When p_i receives a $\text{NEW}(qset, ph\_nb)$ message, its behavior depends on ph_nb.
  - If $ph\_nb > ph\_nb_i$, p_i jumps to the phase ph_nb, adopts the quorum set $qset$ it receives (line 03), and broadcasts its new state (line 04).
  - If $ph\_nb < ph\_nb_i$, p_i discards the message.
  - If $ph\_nb = ph\_nb_i$, p_i and the message are at same phase. In that case, p_i adds $qset$ to its quorum set $quorum\_set_i$. Moreover, if this addition has changed its value, p_i gossips it (line 07).
- The procedure $\mathsf{pres\_inv\&gossip}()$ is invoked in a **when** statement when $quorum\_set_i$ has been modified (line 02 or line 07). It has a reset role and a gossip role.
  - Reset. The first is to preserve the invariant property stated before. To that end, p_i resets $quorum\_set_i$ if the property was about to be violated (lines 13-14). In that case, p_i starts a new phase.
  - Gossip. Then, in all cases, p_i broadcasts the new value of $quorum\_set_i$.
- Finally, the algorithm defines as follows the value returned as the current local output of Ω_k (lines 09-12). The process p_i first considers all the increasing sequences of k process identities the intersection of which with each quorum currently in $quorum\_set_i$ are not empty (lines 09-10). Let us notice that each of these sequences satisfies the invariant property. Then, p_i deterministically selects and returns one of them (e.g., the first in lexicographical order, lines 11-12).

Init: $ph\_nb_i \leftarrow 0$; $quorum\_set_i \leftarrow \{qr_i\}$; broadcast $\text{NEW}(quorum\_set_i, ph\_nb_i)$.

when the value of qr_i **changes**:
(01) $quorum\_set_i \leftarrow quorum\_set_i \cup \{qr_i\}$;
(02) **if** ($quorum\_set_i$ has changed) **then** $\mathsf{pres\_inv\&gossip}()$ **end if**.

when $\text{NEW}(qset, ph\_nb)$ **is received**:
(03) **case** $ph\_nb > ph\_nb_i$ **then** $ph\_nb_i \leftarrow ph\_nb$; $quorum\_set_i \leftarrow qset$;
(04) broadcast $\text{NEW}(quorum\_set_i, ph\_nb_i)$
(05) $ph\_nb < ph\_nb_i$ **then** discard the message
(06) $ph\_nb = ph\_nb_i$ **then** $quorum\_set_i \leftarrow quorum\_set_i \cup qset$;
(07) **if** ($quorum\_set_i$ has changed) **then** $\mathsf{pres\_inv\&gossip}()$ **end if**
(08) **end case**.

when p_i **reads** $leaders_i$:
(09) **let** k_seqs the set of length k increasing sequences of process ids
(10) $\ell_1 < \cdots < \ell_k$ such that $\forall qr \in quorum\_set_i$: $qr \cap \{\ell_1, \ldots, \ell_k\} \neq \emptyset$;
(11) **let** $\ell_1, \ldots, \ell_k$ **be** the first sequence of k_seqs (according to lexicographical order);
(12) $\mathsf{return}(\{\ell_1, \ldots, \ell_k\})$. % local output of Ω_k %

procedure $\mathsf{pres\_inv\&gossip}()$:
(13) **if** ($\nexists\{\ell_1, \ldots, \ell_k\} : \forall qr \in quorum\_set_i : qr \cap \{\ell_1, \ldots, \ell_k\} \neq \emptyset$)
(14) **then** $ph\_nb_i \leftarrow ph\_nb_i + 1$; $quorum\_set_i \leftarrow \{qr_i\}$ **end if**;
(15) broadcast $\text{NEW}(quorum\_set_i, ph\_nb_i)$.

Figure 2: From Π_k to Ω_k (code for p_i)

4.2.2 Proof of the algorithm

As Ω_k is defined by an *eventual* property, let us consider the time instant definition with respect to a run of the algorithm described in Figure 2.

Definition 4 *Let τ be the time instant $\max(\tau_\alpha, \tau_\beta, \tau_\gamma, \tau_\delta)$ where*

1. *From τ_α : all the faulty processes have crashed,*
2. *From τ_β : for each alive process p_i : qr_i contains only correct processes,*
3. *From τ_γ : $\exists\{\ell_1, \ldots, \ell_k\}$ such that, for any alive process p_i, we have $\{\ell_1, \ldots, \ell_k\} \cap qr_i \neq \emptyset$,*
4. *From τ_δ : all the messages $\text{NEW}()$ sent before $\max(\tau_\alpha, \tau_\beta, \tau_\gamma)$ are received and processed.*

Let us notice that τ is well-defined. This follows from the observation that τ_α is well-defined for any run, τ_β and τ_γ are well-defined due to the liveness property and the eventual leadership property of Π_k respectively, and τ_δ is well-defined due to the reliability of the underlying communication network.

Lemma 1 *Let X be the value of the the greatest local variable ph_nb_i at time τ. X is finite and no ph_nb_i variable becomes greater than $X+1$.*

Proof Let us first observe that, as τ is finite and only a finite number of messages can be exchanged in a finite duration, X is finite. The rest of the proof is by contradiction. Let us assume that a process sets its phase number to $X+2$. Let p_i be the first process that does it. As it is the first to proceed to the phase $X+2$, p_i has necessarily increased ph_nb_i to $X+2$ at line 14 (p_i cannot receive a message $\text{NEW}(qset, X+2)$ while it is in phase $X+1$ and proceeds to the phase $X+2$ at line 03). As no process was in the phase $X+1$ at time τ (very definition of X), it follows that all the sets $quorum\_set_j$ sent during the phase $X+1$ contain only quorums qr_x whose value was the local output of Π_k after τ (line 13). Consequently, all the messages $\text{NEW}(qset, X+1)$ received by p_i are such that $qset$ contains only quorums qr_x whose value has been obtained after τ. It then follows from $\tau \geq \tau_\gamma$, that there is a set $\{\ell_1, \ldots, \ell_k\}$ such that $\forall qr \in qset : qr \cap \{\ell_1, \ldots, \ell_k\} \neq \emptyset$. We then conclude that, if the reception of $\text{NEW}(qset, X+1)$ entails the invocation of pres_inv&gossip(), the test of line 13 is false. Hence, ph_nb_i is not increased, which proves the lemma. $\Box_{Lemma\ 1}$

Lemma 2 *There is a finite time after which no message are exchanged.*

Proof The proof follows from the following three observations.

- As the number of processes n is bounded, there is a bounded number of distinct quorums.
- During a phase, no process p_i sends twice the same set of quorums $quorum\_set_i$ (line 02 and 07).
- The number of phases executed by a process is finite.

$\Box_{Lemma\ 2}$

Lemma 3 *The set k_seqs defined at line 09 is never empty, and each of its elements is a non-empty set.*

Proof The proof is by induction. Initially, $quorum\_set_i = \{qr_i\}$, and consequently k_seqs is not empty. Moreover, it follows from the the Intersection property of Π_k qr_i is not empty.

Let us assume that, before modifying $quorum\_set_i$ is modified, k_seqs is not empty and each of its element is a non-empty set. We show the modification of $quorum\_set_i$ keeps these properties. The variable $quorum\_set_i$ can be modified at line 02, line 03, line 07, or line 14.

- $quorum\_set_i$ is modified at line 03. In that case, $quorum\_set_i$ takes the value of $qset$ that, due to the induction assumption, satisfies the property.
- $quorum\_set_i$ is modified at line 14. This case is a reset of $quorum\_set_i$: it is exactly the same as the initialization case. Hence, $quorum\_set_i$ then contains only the non-empty set.
- $quorum\_set_i$ is modified at line 02 or 07. In both cases, the procedure pres_inv&gossip() is invoked. The case where line 14 is executed has been dealt with in the previous item. If the line 14 is not executed, the predicate $\exists\{\ell_1, \ldots, \ell_k\} : \forall qr \in quorum\_set_i : qr \cap \{\ell_1, \ldots, \ell_k\} \neq \emptyset$ is satisfied. But this predicate is exactly the predicate that states that k_seqs is not empty and none of its elements is the empty set (line 12).

$\Box_{Lemma\ 3}$

Lemma 4 $\exists LD = \{\ell_1, \ldots, \ell_k\} : LD \cap \mathcal{C} \neq \emptyset : \exists \tau' \geq \tau$*:* $\forall \tau'' \geq \tau'$*:* $\forall i \in \mathcal{C}$*:* $leaders_i^{\tau''} = LD$.

Proof Let M be the greatest phase number ever attained by a correct process. Due to Lemma 1 this phase number does exist. Moreover, due to the lines 15 and 03, all the correct processes enter the phase M.

During the phase M, each correct process p_i exchanges its quorum set $quorum\_set_i$ each time this set is modified (lines 02 and 07). It follows from the network reliability and the fact that, during a phase, $quorum\_set_i$ can take a bounded number of distinct values, that there is a finite time after which all the correct processes have the same set of quorums in their local variables $quorum\_set_i$ (line 03). Let QS be this set of quorums.

Let τ' be a time after which all the processes p_i are such that $quorum\_set_i = QS$. The first part of the lemma follows from the fact that, after τ', the processes compute deterministically the same set LD of k leaders from the (never changing) same input QS (lines 09-12).

The fact that LD contains a correct process follows from the the liveness property of Π_k (there is a finite time after which each qr_i contains only correct processes), from which we conclude that the quorum set QS contains only quorums made up of correct processes. Due to its very definition, it follows that LD contains at least one correct process. $\Box_{Lemma\ 4}$

Theorem 2 *The algorithm described in Figure 2 is a wait-free quiescent construction of a failure detector of the class Ω_k in $\mathcal{AS}_n[\Pi_k]$.*

Proof The fact that the algorithm constructs a failure detector of the class Ω_k follows from Lemma 3 (validity), and Lemma 4 (eventual leadership). The fact that the algorithm is quiescent follows from Lemma 2. Finally, it is trivially wait-free as there is no wait statement. $\Box_{Theorem\ 2}$

Theorem 3 $\Pi_k \simeq \Sigma_k \times \Omega_k$.

Proof Theorem 1 has proved that $\Sigma_k \times \Omega_k \geq \Pi_k$. Theorem 2 has proved that $\Pi_k \geq \Omega_k$. Finally, (as already noticed), taking taking the output of Π_k as the output of Σ_k proves that $\Pi_k \geq \Sigma_k$. $\Box_{Theorem\ 3}$

5 Π_{n-1} vs $\mathcal{L}$

5.1 The failure detector class $\mathcal{L}$

The failure detector class $\mathcal{L}$ (for loneliness) has been introduced in [11] where it is shown to be the weakest failure detector class that solves the $(n-1)$-set agreement problem in message-passing systems. ([11] also shows that $\mathcal{L}$ is strictly stronger than $\overline{\Omega}_{n-1}$ and strictly weaker than Σ.)

It is defined as follows. Each process p_i is provided with a boolean variable $alone_i$ that it can only read. These variables are such that:

- Stability. There is at least one process whose boolean remains always $false$.
- Loneliness. If only one process is correct, eventually its boolean outputs $true$ forever.

By definition, after a process p_i has crashed (if it ever crashes) its boolean $alone_i$ is set to $false$ and keeps that value forever.

Let us notice that nothing prevents the value of a boolean $alone_i$ to change infinitely often (as long as the corresponding process p_i is neither the one whose boolean remains always false, nor the only correct process in the the case where all the other process crash).

5.2 From Π_{n-1} to $\mathcal{L}$

The algorithm that constructs a failure detector of the class $\mathcal{L}$ from any failure detector of the class Π_{n-1} is described in Figure 3. It is pretty simple: the boolean of a process p_i becomes true (and remains true forever) only if the quorum of that process contains only its own identity. (A similar construction is described in [11] to show that Σ is stronger than $\mathcal{L}$.)

Init: $alone_i \leftarrow false.$
when $qr_i = \{i\}$: $alone_i \leftarrow true.$

Figure 3: From Σ_{n-1} to $\mathcal{L}$ (code for p_i)

Theorem 4 *The algorithm described in Figure 3 builds a failure detector of the class $\mathcal{L}$ in $\mathcal{AS}_n[\Sigma_{n-1}]$.*

Proof The Loneliness property of $\mathcal{L}$ follows from a simple observation. If a single process p_i is correct, it follows from the Liveness property of Π_{n-1} that eventually $qr_i = \{i\}$. When this occurs $alone_i$ is set to true and remains true forever.

The proof of the Stability property of $\mathcal{L}$ is by contradiction. Let us assume that all the boolean variables $alone_i$ are set to true. Due to the initialization, this means that, for each p_i, we had at some time $qr_i = \{i\}$. But this violates the Intersection property of Σ_{n-1}. Consequently, there is at least one process whose boolean variable remains always false. $\Box_{Theorem\ 4}$

The following corollary is an immediate consequence of the fact that $\Pi_{n-1} = \Sigma_{n-1} \times \Omega n - 1$.

Corollary 1 *The algorithm described in Figure 3 builds a failure detector of the class $\mathcal{L}$ in $\mathcal{AS}_n[\Pi_{n-1}]$.*

5.3 From $\mathcal{L}$ to Π_{n-1}

The algorithm that constructs a failure detector of the class Π_{n-1} from any failure detector of the class $\mathcal{L}$ is described in Figure 4. It is very simple. Each process p_i periodically sends ALIVE(i) messages, processes the messages it receives, and set qr_i to $\{i\}$ when $alone_i$ becomes true (then, qr_i is no longer modified).

Theorem 5 *The algorithm described in Figure 4 is a wait-free construction of a failure detector of the class Π_{n-1} in $\mathcal{AS}_n[\mathcal{L}]$.*

Init: $qr_i \leftarrow \{i, j\}$ where $j \neq i$.

Task $T1$: **repeat periodically** broadcast ALIVE(i) **end repeat**.

Task $T2$: **when** $alone_i$ **becomes** $true$: $qr_i \leftarrow \{i\}$.
when ALIVE(j) **is received**: **if** $\big((i \neq j) \wedge (|qr_i| \neq 1)\big)$ **then** $qr_i \leftarrow \{i, j\}$ **end if**.

Figure 4: From $\mathcal{L}$ to Π_{n-1} (code for p_i)

Proof The proof considers each property of Π_{n-1} separately.

Proof of the Intersection property. As $k = n-1$, we have to prove that $\forall\{\tau_1, \dots, \tau_n\} : \exists i, j : 1 \leq i \neq j \leq n : (qr_i^{\tau_i} \cap qr_j^{\tau_j} \neq \emptyset)$. Due to the Stability property of $\mathcal{L}$, there is at least one process (say p_i) such that $alone_i$ never becomes $true$. So, until p_i crashes (if it ever crashes), we have $|qr_i| = 2$. Consequently, there is always a process p_j such that $qr_i = \{i, j\}$, from which it follows that there is always a process p_j (not necessarily always the same) such that at any time $qr_i \cap qr_j \neq \emptyset$, which proves the property until p_i crashes. After p_i has crashed (if it does), the Intersection property is trivially satisfied.

Proof of the Liveness property. Let p_i be a correct process. We consider two cases.

- The boolean $alone_i$ takes (at least once) the value $true$. In that case, we will have $qr_i = \{i\}$. Then, qr_i remains forever equal to $\{i\}$, and the Liveness property is satisfied.
- The boolean $alone_i$ never takes the value $true$, and consequently we will never have $qr_i = \{i\}$. In that case, there are other correct processes (at least one). As, after some finite time, there are only correct processes, p_i will receive infinitely often messages ALIVE(j) from each of these correct processes p_j (and it will receive messages only from them). It follows that, after some time, qr_i contains only ids of correct processes.

Proof of the Eventual leadership property. We have to prove that $\exists \tau : \exists LD = \{\ell_1, \dots, \ell_{n-1}\} : \forall \tau' \geq \tau : \forall i : qr_i^{\tau'} \cap LD \neq \emptyset$. Let us recall that any boolean (but one) can flip infinitely often between $false$ and $true$. Let τ be the time after which no more boolean moves from $false$ to $true$ for the first time. Let $Z = \{i | \exists \tau : alone_i^{\tau} = true\}$. It follows from the definition of $\mathcal{L}$ that $0 \leq |Z| \leq n-1$. We consider two cases.

- $|Z| = n-1$. Let $Z = \{\ell_1, \dots, \ell_{n-1}\}$ and take $LD = Z$. We show that, in that case, after τ, we always have $\forall i : LD \cap qr_i \neq \emptyset$. This is trivial for any process p_{ℓ_x}, $1 \leq x \leq n-1$, as we always have $\ell_x \in qr_{\ell_x}$. Let us now consider the process p_{ℓ_n} such that $alone_{\ell_n}$ remains always equal to $false$ (due to definition of $\mathcal{L}$, p_{ℓ_n} does exist). Due to the algorithm of Figure 4, the process p_{ℓ_n} is such that we always have $|qr_{\ell_n}| = 2$. Consequently, the predicate $qr_{\ell_n} \cap LD \neq \emptyset$ is always satisfied, which completes the proof of the case.
- $|Z| < n-1$. Let $|Z| = z$. Let us recall that each process p_i in Z is such that after some finite time we always have $qr_i = \{i\}$. In that case, let us add $(n-1) - z$ processes to Z in order to obtain a set LD of $(n-1)$ processes. Due to the definition of Z and the algorithm of Figure 4, it follows that the process (say p_{ℓ_n}) that is not in LD is such that $|qr_{\ell_n}| = 2$. Consequently (as in the previous item) the predicate $qr_{\ell_n} \cap LD \neq \emptyset$ is always satisfied. Hence, the set LD satisfies the Eventual leadership property, which completes the proof of the theorem.

$\Box_{Theorem\ 5}$

5.4 Σ_{n-1}, $\mathcal{L}$ and Ω_{n-1}

Theorem 6 $\Sigma_{n-1} \simeq \mathcal{L} \simeq \Pi_{n-1} \simeq \Sigma_{n-1} \times \Omega_{n-1}$.

Proof The proof follows from Theorem 4 (that builds $\mathcal{L}$ from Σ_{n-1}), Theorem 5 (that builds Π_{n-1} from $\mathcal{L}$), and Theorem 3 (that builds $\Sigma_{n-1} \times \Omega_{n-1}$ from Π_{n-1}), and the fact that Σ_{n-1} is trivially obtained from $\Sigma_{n-1} \times \Omega_{n-1}$. $\Box_{Theorem\ 6}$

This theorem generalizes a result of [9] where it is shown that $\Sigma_1 \simeq \Sigma_1 \times \Omega_1$ in systems made up $n = 2$ processes. The following corollaries are an immediate consequence of the previous theorem and the definition of Σ_k. The second one generalizes a result of [11] that (expressed with our notations) states $\Sigma_1 \succ \mathcal{L} \succ \overline{\Omega}_{n-1}$.

Corollary 2 Σ_{n-1} *is stronger than* Ω_{n-1}.

Corollary 3 $\Sigma_1 \succ \Sigma_2 \succ \dots \succ \Sigma_{n-2} \succ \Sigma_{n-1} \simeq \mathcal{L}$.

6 A Σ_{n-1}-based $(n-1)$-set agreement algorithm

An $\mathcal{L}$-based $(n-1)$-set agreement algorithm is presented in [11]. Hence, the stacking of this algorithm on top of the algorithm described in Figure 4 (that builds Π_{n-1}, i.e., Σ_{n-1}, in $\mathcal{AS}_n[\mathcal{L}]$), supplies a Σ_{n-1}-based $(n-1)$-set agreement algorithm. This Section describes a $(n-1)$-set agreement algorithm that is directly built on top of Σ_{n-1} and consequently saves the construction of $\mathcal{L}$ when one is provided with a failure detector of the class Π_{n-1}.

6.1 The algorithm

The code of the algorithm for a process p_i is described in Figure 5. The local variable est_i contains p_i's current estimate of the decision value, while $qsize_i$ contains a quorum size, namely, the size of smallest quorum that allowed computing the current value of est_i.

The processes proceed in n asynchronous rounds. At the end of the last round, p_i returns (decides) the current value of est_i (line 09). During a round r, a process p_i first broadcasts it current state (the pair $(qsize_i, est_i)$) and waits for the current states of the processes in its current quorum qr_i (lines 03-04). Then, considering these states $(qsize, est)$ plus its local state, p_i selects the smallest one according to their lexicographical ordering[4] (line 06). Finally, p_i updates $qsize_i$ and est_i (line 07). The local estimate est_i is updated to the estimate value est_x of the processes p_x of $q = qr_i \cup \{i\}$ such that $qsize_x$ is the smallest; $qsize_i$ is set to $\min(qsize_x, |q|)$ to take into account the size of the quorum that allowed computing est_i (line 07).

Function set_agreement$_{n-1}$ (v_i):
(01) $est_i \leftarrow v_i$; $qsize_i \leftarrow n$;
(02) **for** r_i **from** 1 to n **do**
(03) broadcast PROPOSE$(r_i, qsize_i, est_i)$;
(04) **wait until** $($ PROPOSE$(r_i, -, -)$ **received from all the processes in** $qr_i)$;
(05) **let** q **be** $\{i\}\cup$ the quorum qr_i that allowed the **wait** statement to terminate;
(06) **let** $(qsize, est)$ **be** the smallest pair (lex. order) rec. from the processes $\in q$;
(07) $qsize_i \leftarrow \min(qsize, |q|)$; $est_i \leftarrow est$
(08) **end for**;
(09) return(est_i).

Figure 5: Σ_{n-1}-based $(n-1)$-set algorithm (code for p_i)

6.2 Proof of the algorithm

Notation 1 *Let est_i^r denote the value of est_i at the end of the round r (that is the value of est_i at the beginning of the round $(r+1)$ if p_i starts that round). Let $EST[r] = \bigcup_i \{est_i^r\}$.*

Lemma 5 *Let r be a round, $1 \leq r \leq n$. At the end of r, (i) $|EST[r]| \leq (n-1)$, or (ii) the process p_i that has the greatest pair $(qsize_i, est_i)$ at the beginning of the round r, is such that $qsize_i = 1$ at the end of the round r.*

Proof Let us consider a round r, and assume that Item (i) is not satisfied, i.e., we have $|EST[r]| = n$. The proof shows that Item (ii) is then satisfied. Let p_i be a process with the highest $(qsize, est)$ pair (according to lexicographical order). As $|EST[r]| = n$, all the estimate values are different at the end of r, from which follows that the process p_i is unique.

Let us first observe that no other process p_j can adopt the value est_i of p_i. This is because when p_j executes line 05 we have $j \in q$ and the pair $(qsize_i, est_i)$ is the highest according to lexicographical order, from which we conclude that p_j cannot select it at line 06.

Let us now consider p_i. If it receives at line 04 messages from other processes (i.e., $qr_i \neq \{i\}$), it adopts one of these pairs to define its new value of $qsize_i$ and est_i. We then have $|EST[r]| < n$ which contradicts the assumption stating that Item (i) is not satisfied. Consequently, this case cannot occur. If, at line 04, p_i receives a message only from itself, we then have $qr_i = \{i\}$, i.e., $|q| = 1$ at line 05. In that case, $qsize_i$ is set to 1 at line 07 which concludes the proof of the lemma. $\Box_{Lemma\ 5}$

Lemma 6 *If, during a round r, $2 \leq r \leq n$, a process p_i sets $qsize_i$ to 1 due to another process (i.e., while $|q| \neq 1$ at line 07), then two processes have the same estimate value at the end of that round.*

Proof Let $ONE[\rho]$ be the set of processes p_x such that $qsize_x = 1$ at the end of the round ρ, and $EST\_ONE[\rho]$ be the set of their estimates at the end of ρ. The definition $ONE[\rho]$ is extended as follows for the processes that crash. If a process p_x crashes after it has been added to $ONE[\rho]$, it is also added to $ONE[\rho']$ for all ρ' such that $\rho \leq \rho' \leq n$. We consequently have $ONE[\rho] \subseteq ONE[\rho+1]$.

[4] Recall that this order is defined as follows: $(q1, est1) < (q2, est2) \stackrel{\text{def}}{=} \big((q1 < q2) \vee (q1 = q2 \wedge est1 < est2)\big)$.

Let $r > 1$ be a round. Let us consider the processes p_y that are in $ONE[r-1]$ and execute the round r. As all these processes p_x are such that $qsize_x = 1$, some of them can adopt the estimate value of other processes but those processes belong to $ONE[r-1]$. The important point is that the set of their estimate values remains the same or decreases during the round r.

Let us now consider the process p_i defined in the lemma assumption. It is such that $i \notin ONE[r-1]$ and $i \in ONE[r]$ (it is during r that p_i set $qsize_i$ to 1 while $|q| \neq 1$). Consequently, p_i has adopted an estimate est associated with an integer $qsize = 1$. It follows that $est_i^r \in EST\_ONE[r-1]$.

It follows from the previous observations that $|ONE[r-1]| < |ONE[r]|$ and $EST\_ONE[r-1] = EST\_ONE[r]$, from which we conclude that two processes of $ONE[r]$ have the same estimate value. $\Box_{Lemma\ 6}$

Theorem 7 *The Σ_{n-1}-based algorithm described in Figure 5 solves the $(n-1)$-set agreement in a wait-free environment.*

Proof The validity property of the k-set agreement problem follows from the initialization of the local variables est_i and the fact that, when it is updated to a new value, such a variable can only take the value of one of the estimates values (lines 03, 06 and 07).

The termination property consists in showing that no correct process can block forever at line 04. The proof is by contradiction. Let r be the first round during which a correct process blocks forever at line 04. As no correct process blocks forever during a round $r' < r$, it follows that every correct process broadcasts a message PROPOSE$(r_i, -, -)$ when it starts the round r. Moreover, due to the liveness property of Σ_{n-1}, there is a finite time after which qr_i contains only correct processes. If follows from these observations that there is a finite time after which p_i has received a round r message from all the processes in qr_i, and consequently no correct process can block forever at round r which contradicts the definition of the round r. Hence, all the correct processes decide.

The proof of the agreement property (at most $(n-1)$ distinct values are decided) is by contradiction. Let us assume that n distinct values are decided. Hence, each process executes the n rounds and decides at the end of the round n, which means that $|EST[n]| = n$. The proof is a consequence of the following items.

1. It follows from $EST[r+1] \subseteq EST[r]$ and $|EST[n]| = n$, that $\forall\, r : 1 \leq r < n : \;\; [EST[r]| = n$.
2. Initially, all the variables $qsize_i$ are equal to n.
3. Due to the lines 06-07, once a process p_x has updated $qsize_x$ to 1, $qsize_x$ keeps that value forever.
4. Let us consider the case where there is at least one process p_j such that $qsize_j > 1$ at the beginning of a round r. As $|[EST[r]| = n$ (Item 1), Item (i) of Lemma 5 does not apply. So, it follows from Item (ii) of this lemma that, the process p_j, the $(qsize_j > 1, est_j)$ of which is the greatest at the beginning of r, is such that $qsize_j = 1$ at the end of that round.
5. It follows from the previous items 2,3 and 4 that all the processes p_i are such that $qsize_i = 1$ at the end of the round $r = n$.

Let us notice that, as there are n distinct values at the end of the round n ($|[EST[r]| = n$), it follows from Lemma 6 that the update of $qsize_i$ to 1 by p_i is due to the fact that $q = qr_i \cup \{i\}$ with $|q| = 1$ when p_i has executed the lines 04-07 during some round r (otherwise, due to Lemma 6, we would have $|[EST[r]| < n$). Consequently, for each process p_i, there is a time τ_i such that $qr_i^{\tau_i} = \{i\}$, which contradicts the intersection property of Σ_1 (in any set of n quorums, two of them have to intersect), and concludes the proof of the k-set agreement property. $\Box_{Theorem\ 7}$

7 Necessity of Σ_k to solve k-set agreement

This section shows that Σ_k is necessary to solve the k-set agreement problem as soon as we are looking for a failure detector-based solution. To that end, given any algorithm A that solves the k-set agreement problem with the help of a failure detector $\mathcal{D}$, we provide an algorithm that emulates the output of Σ_k. This means that it is possible to build a failure detector of the class Σ_k from any failure detector $\mathcal{D}$ that can solve the k-set agreement problem (according to the usual terminology, Σ_k can be *extracted from* the $\mathcal{D}$-based algorithm A). The output of Σ_k at p_i is kept in qr_i.

Interestingly enough, and in addition of being more general, the proposed construction (Figure 6) provides us with a proof of the necessity of Σ_1 to solve the consensus problem that is simpler that the one described in [9].

Underlying principle As in [11], the proposed extraction algorithm does not rely on the asynchronous impossibility of a problem. Its design principle is the following. Each process p_i participates in several runs of A. Let $R_{\{i\}}$ denote a run of A in which only the process p_i participates, $R_{\{i,j\}}$ $(i \neq j)$ a run of A in which only the processes p_i and p_j participate, etc., and $R_{\{1,2,\ldots,n\}}$ a run of A in which all the processes participate. This means that in a run denoted R_Q only the processes of Q take steps, and each process of Q either decides, blocks forever or crashes[5]. So, the extraction algorithm uses $2^n - 1$ runs of A. Let us observe that, due to asynchrony and the fact that any number of processes can crash ("wait-free" environment), any prefix of any of these runs can occur in a given execution.

[5]As the processes that are not in Q do not participate, the messages sent by the processes of Q to these processes are never received. Alternatively, as in [11], we could say that the processes of Q "omit" sending messages to the processes that are not in Q.

The algorithm The algorithm executed by each process p_i is described in Figure 6. Each process manages two local variables: a set of sets denoted S_i and a queue denoted $queue_i$. The aim of S_i is to contain all the sets Q such that p_i decides in the run R_Q (Task $T1$), while $queue_i$ is managed as the queue with the same name in Figure 1 (task $T2$ and first **when** statement of $T3$). The important point point is here that the correct processes eventually appear before the faulty processes in $queue_i$.

The idea is to select a set of S_i as the current output of Σ_k. As we will see in the proof, any $(k+1)$ sets of S_i are such that two of them do intersect which will supply the intersection property. The main issue is to ensure the liveness property of Σ_k (namely, eventually the set qr_i associated with p_i contains only correct processes), while preserving the intersection property. This is done as follows with the help of $queue_i$. The current output of Σ_k is the set (quorum) of S_i that appears as being the "first" in $queue_i$. The formal definition of "first set of S_i wrt $queue_i$" is stated in the task $T3$. To make it easy to understand let us consider the following example. Let $S_i = \{\{3,4,9\},\{2,3,8\},\{4,7\}\}$, and $queue_i = < 4,8,3,2,7,5,9,\cdots >$. The set $F = \{2,3,8\}$ is the first set of S_i with respect to $queue_i$ because each of the other sets $\{3,4,9\}$ and $\{4,7\}$ includes an element (9 and 7, respectively) that appears in $queue_i$ after the elements of F. (In case several sets are "first", any of them can be selected).

Init: $S_i \leftarrow \{\{1\ldots,n\}\}$; $queue_i \leftarrow < 1,\ldots,n >$;
  for each $Q \in (2^{\Pi} \setminus \{\emptyset,\{1,\ldots,n\}\})$ **such that** $(i \in Q)$ **do**
    let A_Q denote the $\mathcal{D}$-based instance of A in which participate only the processes of Q;
    p_i proposes i to A_Q **end for**.

Task $T1$: **when** p_i decides in the instance of A in which participate only the processes of Q: $S_i \leftarrow S_i \cup \{Q\}$.

Task $T2$: **repeat periodically** broadcast ALIVE(i) **end_repeat**.

Task $T3$: **when** ALIVE (j) **is received**: suppress j from $queue_i$; enqueue j at the head of $queue_i$.
  when p_i reads qr_i: **let** $m = \min_{Q \in S_i}(\max_{x\in Q}(rank[x]))$ where $rank[x]$ denotes the rank of x in $queue_i$;
    return (a set Q such that $\max_{x\in Q}(rank[x]) = m$).

Figure 6: Extracting Σ_k from a k-set agreement failure detector-based algorithm A

Remark Initially S_i contains the set $\{1,\ldots,n\}$. As only sets of processes can be added to S_i (task $T1$), S_i is never empty. Moreover, it is not necessary to launch a run in which all the processes participate. This is because, as the $\mathcal{D}$-based k-set agreement algorithm A is correct, it follows that all the correct processes decide in that run $R_{\{1,\ldots,n\}}$. This case is directly taken into account in the initialization of S_i (thereby saving the run $R_{\{1,\ldots,n\}}$).

Theorem 8 *Given any algorithm A that solves the k-set agreement problem with the help of a failure detector $\mathcal{D}$, The algorithm described in Figure 6 is a wait-free construction of a failure detector of the class Σ_k.*

Proof The Intersection property of Σ_k is proved by contradiction. Let us first notice that a set qr_i returned to a process p_i is a set Q of S_i. Let us assume that there are $k+1$ subsets of processes $Q_1,\ldots,Q_{k+1}$ that (1) $\forall x : 1 \le x \le k+1 : Q_x \in \bigcup_{1\le i\le n} S_i$, and (2) $\forall x,y : 1 \le x \ne y \le k+1 : Q_x \cap Q_y = \emptyset$. (pairwise independence). The item (1) means that Q_x can be returned as the value of qr_i by a process p_i.

Let $Q = Q_1 \cup \ldots \cup Q_{k+1}$. Let R be the run of A in which (1) only the processes of Q participate, and (2) for each x, $1 \le x \le k+1$, the processes of Q_x behave exactly as in R_{Q_x} (as defined in the **Init** part of Figure 6). Due to the second item, in R, the processes in Q_x, $1 \le j \le k+1$, that decide do decide as in R_{Q_x}. It follows that, even if the processes in each Q_x would decide the same value, up to $k+1$ different values could be decided. This contradicts the fact that A solves the k-set agreement in the run R, from which we conclude that $\exists x,y : 1 \le x \ne y \le k+1 : Q_x \cap Q_y \ne \emptyset$ which proves the Intersection property of Σ_k.

As far as the Liveness property, let us consider the run of A in which the set of participating processes is exactly $\mathcal{C}$ (the set of correct processes). Due to the termination property of A, every correct process does terminate in that instance. Consequently, in the extraction algorithm, the variable S_i of each correct process p_i eventually contains the set $\mathcal{C}$.

Moreover, after some finite time, each correct process p_i receives ALIVE(j) messages only from correct processes. This means that, for each correct process p_i, all the correct processes eventually precede the faulty processes in $queue_i$. Due to the definition of "first set of S_i wrt $queue_i$" stated in the task $T3$, and the fact that $\mathcal{C} \in S_i$, it follows that the quorum Q selected by the task $T3$ is such that $Q \subseteq \mathcal{C}$, which proves the liveness property of Σ_k. $\Box_{Theorem\ 8}$

8 Concluding remark

This paper has addressed the question of the weakest failure detector class to solve the k-set agreement problem in asynchronous message-passing systems prone to any number of process crashes. It has proposed Π_k as a candidate for the corresponding failure detector class, and has shown that (1) Π_1 and Π_{n-1} are indeed the weakest classes for $k=1$ and $k=n-1$, respectively, and (2) Σ_k is a necessary requirement for any k. Although it seems a posteriori simple, finding a single parameterized formulation for $\Sigma_1 \times \Omega_1$ and $\mathcal{L}$ was not a priori evident. The remaining question is now: is Π_k the end of the road or has it to be made stronger in order to be the weakest class when $1 < k < n$?

References

[1] Afek Y., Gafni E., Rajsbaum S., Raynal M. and Travers C., Simultaneous consensus tasks: a tighter characterization of set consensus. *Proc. 8th Int'l Conference on Distributed Computing and Networking (ICDCN'06)*, Springer Verlag LNCS #4308, pp. 331-341, 2006.

[2] Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on Theory of Computation (STOC'93)*, San Diego (CA), pp. 91-100, 1993.

[3] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, 1996.

[4] Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[5] Chaudhuri S., More *Choices* Allow More *Faults:* Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation*, 105:132-158, 1993.

[6] Chaudhuri S., Herlihy M., Lynch N. and Tuttle M., Tight Bounds for k-Set Agreement. *Journal of the ACM*, 47(5):912-943, 2000.

[7] Chen W., Zhang J., Chen Y. and Liu X., Weakening failure detectors for k-set agreement via the partition approach. *Proc. 21th Int'l Symposium on Distributed Computing (DISC'07)*, Springer-Verlag LNCS #4731, pp. 123-138, 2007.

[8] Delporte-Gallet C., Fauconnier H. and Guerraoui R., Sharing is harder than agreeing. *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*, ACM Press, pp. 85-94, Toronto (Canada), 2008.

[9] Delporte-Gallet C., Fauconnier H. and Guerraoui R., Shared Memory vs Message Passing. *Technical Report2003-77*, EPFL Lausanne, 2003.

[10] Delporte-Gallet C., Fauconnier H., Guerraoui R., Hadzilacos V., Kouznetsov P. and Toueg S., The weakest failure detectors to solve certain fundamental problems in distributed computing. *Proc. 23th ACM Symposium on Principles of Distributed Computing (PODC'04)*, ACM Press, pp. 338-346, 2004.

[11] Delporte-Gallet C., Fauconnier H., Guerraoui R. and Tielmann A., The Weakest Failure Detector for Message Passing Set-Agreement. *Proc. 22th Int'l Symposium on Distributed Computing (DISC'08)*, Springer-Verlag LNCS #5218, pp. 109-120, 2008.

[12] Delporte-Gallet C., Fauconnier H., Guerraoui R. and Tielmann A., The disagreement power of an adversary. *Proc. 28th ACM Symposium on Principles of Distributed Computing (PODC'09)*, ACM Press, 2009.

[13] Fernandez Anta A., Rajsbaum S. and Travers C., Weakest failure detectors with an edge-laying simulation. *Proc. 28th ACM Symposium on Principles of Distributed Computing (PODC'09)*, ACM Press, 2009.

[14] Gafni E. and Kuznetzov P., The weakest failure detector for solving k-set agreeemnt. *Proc. 28th ACM Symposium on Principles of Distributed Computing (PODC'09)*, ACM Press, 2009.

[15] Guerraoui R. and Kouznetsov P., Failure detectors as types boosters. *Distributed Computing*, 20:343-358, 2008.

[16] Herlihy M.P. and Penso L. D., Tight Bounds for k-Set Agreement with Limited Scope Accuracy Failure Detectors. *Distributed Computing*, 18(2): 157-166, 2005.

[17] Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923,, 1999.

[18] Lo W.-K. and Hadzilacos V., Using failure detectors to solve consensus in asynchronous shared-memory systems. *Proc. 8th Int'l Workshop on Distributed Algorithms (WDAG'94, now DISC)*, Springer-Verlag LNCS #857, pp. 280-295, 1994.

[19] Manabe Y., Baldoni R., Raynal M. and Aoyagia S., K-arbiter: a safe and general scheme for h-out of-k mutual exclusion. *Theoretical Computer Science*, 193(1-2): 97-112, 1998.

[20] Mostéfaoui A., Rajsbaum S., Raynal M. and Travers C., On the Computability Power and the Robustness of Set Agreement-oriented Failure Detector Classes. *Distributed Computing*, 21(3):201-222, 2008.

[21] Mostéfaoui A. and Raynal M., k-Set Agreement with Limited Accuracy Failure Detectors. *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 143-152, 2000.

[22] Mostéfaoui A. and Raynal M., Randomized Set Agreement. *Proc. 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA'01)*, ACM Press, pp. 291-297, Hersonissos (Crete), 2001.

[23] Neiger G., Failure Detectors and the Wait-free Hierarchy. *14th ACM Symposium on Principles of Distributed Computing (PODC'95)*, ACM Press, pp. 100-109, Las Vegas -NV), 1995.

[24] Raynal M. and Travers C., In search of the holy grail: looking for the weakest failure detector for wait-free set agreement. *Proc. 10th Int'l Conference On Principles Of Distributed Systems (OPODIS'06)*, Springer-Verlag LNCS #4305, pp. 1-17, 2006.

[25] Raynal M. and Travers C., Synchronous Set Agreement: a concise guided tour (including a new algorithm and a list of open problems). *Proc. 12th IEEE Pacific Rim Int'l Symposium on Dependable Computing (PRDC'06)*, IEEE Computer Press, pp. 267-274, Riverside (CA), 2006.

[26] Raynal M., K-anti-Omega. *Rump Session at 26th ACM Symposium on Principles of Distributed Computing (PODC'07)*, 2007.

[27] Saks M. and Zaharoglou F., Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.

[28] Zielinski P., Anti-Omega: the weakest failure detector for set agreement. *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*, ACM Press, pp. 55-64, Toronto (Canada), 2008.