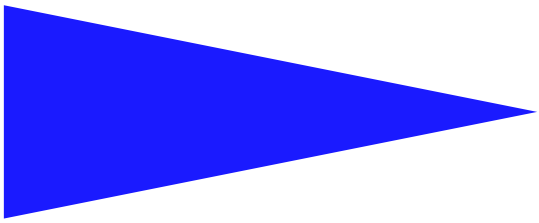
Irreducibility and Additivity of Set Agreement-oriented Failure Detector Classes

Achour Mostefaoui, Sergio Rajsbaum, Michel Raynal, Corentin Travers

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1758



IRREDUCIBILITY AND ADDITIVITY OF SET AGREEMENT-ORIENTED FAILURE DETECTOR CLASSES

A. MOSTEFAUI S. RAJSBAUM M. RAYNAL C. TRAVERS

ISSN 1166-8687

CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

Irreducibility and Additivity of Set Agreement-oriented Failure Detector Classes

A. Mostefaui* S. Rajsbaum** M. Raynal*** C. Travers****

Systèmes communicants

Publication interne n ° 1758 — Novembre 2005 — 24 pages

Abstract: Solving agreement problems (such as consensus and k-set agreement) in asynchronous distributed systems prone to process failures has been shown to be impossible. To circumvent this impossibility, distributed oracles (also called unreliable failure detectors) have been introduced. A failure detector provides information on failures, and a failure detector class is defined by a set of abstract properties that encapsulate (and hide) synchrony assumptions. Some failure detector classes have been shown to be the weakest to solve some agreement problems (e.g., Ω is the weakest class of failure detectors that allow solving the;consensus problem in asynchronous systems where a majority of processes do not crash).

This paper considers several failure detector classes and focuses on their additivity or their irreducibility. It mainly investigates two families of failure detector classes (denoted $\diamond\mathcal{S}_x$ and $\diamond\phi^y$, $0 \leq x, y \leq n$), shows that they can be "added" to provide a failure detector of the class Ω^z (a generalization of Ω). It also characterizes the power of such an "addition", namely, $\diamond\mathcal{S}_x + \diamond\phi^y \rightsquigarrow \Omega^z \Leftrightarrow x + y + z > t + 1$, where t is the maximum number of processes that can crash in a run. As an example, the paper shows that, while $\diamond\mathcal{S}_t$ allows solving 2-set agreement (and not consensus) and $\diamond\phi^1$ allows solving t-set agreement (but not $(t-1)$-set agreement), their "addition" allows solving consensus. More generally, the paper studies the failure detector classes $\diamond\mathcal{S}_x$, $\diamond\phi^y$ and Ω^z, and shows which reductions among these classes are possible and which are not. The paper presents also an Ω^k-based k-set set agreement protocol. In that sense, it can be seen as a step toward the characterization of the weakest failure detector class that allows solving the k-set agreement problem.

Key-words: Asynchronous system, Distributed algorithm, Eventual leader, Fault-tolerance, Limited scope accuracy, Process crash, Reduction algorithm, Scalability, Set agreement, Unreliable failure detector.

(Résumé : tsvp)

* IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France achour@irisa.fr
** Instituto de Matemáticas, UNAM, D. F. 04510, Mexico rajsbaum@matem.unam.mx
*** IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France, raynal@irisa.fr
**** IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France, travers@irisa.fr

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Sur la composition de classes de détecteurs de fautes

Résumé : Ce rapport étudie la composition de classes de détecteurs de fautes. Il montre aussi que certaines classes ne peuvent être composées pour donner des détecteurs d'une puissance supérieure.

Mots clés : Systèmes répartis asynchrones, Tolérance aux fautes, Crash de processus, Leader inéluctable, Classes de détecteurs de fautes, Réduction, Composition.

1 Introduction

Context of the work: failure detectors for agreement problems *Consensus* is one of the most fundamental problem in fault-tolerant distributed computing: each process proposes a value, and every non-faulty process must decide a value (termination) such that no two different values are decided (agreement) and the decided value is a proposed value (validity). Despite the simplicity of its definition and its use as a basic building block to solve distributed agreement problems, consensus cannot be solved in asynchronous system where even a single process can crash [8].

Several approaches have been investigated to circumvent this impossibility result. One of them is the failure detector approach [3, 22]. It consists in equipping the underlying system with a distributed oracle that provides each process with (possibly incorrect) hints on process failures. According to the type and the quality of the hints, several classes of failure detectors can be defined. As far as consensus is concerned, two classes are particularly important.

- The class of *leader* failure detectors [2] (denoted Ω). This class includes all the failure detectors that continuously output at each process a process identity such that, after some time, all the correct processes are provided with the same identity that is the identity of a correct process (eventual leadership). Before that time, different processes can be provided with distinct leaders (that can also change over time), and there is no way for the processes to know when this anarchy period is over. Ω-based asynchronous consensus protocols can be found in [9, 14, 20][1].
- The class of *eventually strong* failure detectors [3] (denoted $\diamond\mathcal{S}$). A failure detector of that class provides each process with a set of suspected processes such that this set eventually includes all the crashed processes (strong completeness) and there is a correct process p and a time after which no set contains the identity of p (eventual strong accuracy). $\diamond\mathcal{S}$-based asynchronous consensus protocols can be found in [3, 9, 18, 24].

Two important results are associated with these classes. First, they are equivalent (which means that it is possible, from any failure detector of any of these classes, to build a failure detector of the other class) [2, 5, 17]. Second, as far information on failures is concerned, they are the weakest class of failure detectors that allow solving consensus in asynchronous systems where a majority of processes are correct [2].

The *k-set agreement* problem relaxes the consensus requirement to allow up to k different values to be decided [4] (consensus is 1-set agreement). This problem is solvable in asynchronous system despite up to $k-1$ process crash failures, but has been shown to be impossible to solve as soon as k or more processes can crash [1, 12, 23].

The failure detector class $\diamond\mathcal{S}$ has been weakened in [19, 25] to address this problem. While the scope of the accuracy property of $\diamond\mathcal{S}$ spans the whole system (there is a correct process that, after some time, is not suspected by any process), the class $\diamond\mathcal{S}_x$ is defined by the same completeness property and a limited scope accuracy property, namely, there is a correct process that, after some time, is not suspected by x processes. It is easy to see that $\diamond\mathcal{S}_n$ (where n is the total number of processes) is $\diamond\mathcal{S}$, while $\diamond\mathcal{S}_1$ provides no information on failures. Moreover, $\diamond\mathcal{S}_{x+1} \subseteq \diamond\mathcal{S}_x$. It has been shown that, when we consider the family $(\diamond\mathcal{S}_x)_{1\leq x\leq n}$ of failure detectors, $\diamond\mathcal{S}_x$ is the weakest class that allows solving k-set agreement in asynchronous systems for $k = t - x + 2$ (where t is an upper bound on the number of processes that can crash) [11] (message-passing systems must also satisfy the additional constraint of a majority of correct processes, $t < n/2$). The class $\mathcal{S}_x$ of failure detectors is a subset of $\diamond\mathcal{S}_x$. It has a the same completeness property but a stronger accuracy property: it requires that, from the very beginning, there is a subset of x processes that a never suspect one correct process.

A new family of failure detectors (denoted $(\phi^y)_{0\leq y\leq n}$), has recently been introduced in [16] (where it is used in conjunction with conditions [15] to solve set agreement problems). A failure detector of the class ϕ^y provides the processes with a query primitive that has a parameter (a set X of processes) and returns a boolean answer. The invocation QUERY(X) by a process returns systematically *true* (resp., *false*) when $0 \leq |X| \leq t - y$, i.e., when the set is too small (resp., $|X| > t$, i.e., when the set is too big). When $t - y < |X| \leq t$ (the set has then an appropriate size) QUERY(X) returns *true* only if all the processes in X have crashed; moreover, if all the processes of X have crashed and a process repeatedly issues QUERY(X), it

[1] It is important to notice that the first version of the leader-based Paxos protocol dates back to 1989, i.e.. before the Ω formalism was introduced.

eventually obtains the answer *true*. We have $\phi^{y+1} \subseteq \phi^y$. Moreover, ϕ^0 provides no information on failures, while, $\forall\ y \geq t$, ϕ^y is equivalent to a perfect failure detector (one that never does a mistake [3]). Φ^y is a subclass of ϕ^y that additionally requires the sets passed as query parameters to satisfy a containement property (i.e., any two such sets $X1$ and $X2$ need to be such that $X1 \subseteq X2$ or $X2 \subseteq X1$). It is shown in [16], that, in shared memory systems, Φ^y is the weakest failure detector class of the family $(\Phi^y)_{0 \leq y \leq t}$ that allows solving asynchronous k-set agreement with $k = t - y + 1$.

The family of failure detector classes $(\Omega^z)_{1 \leq z \leq n}$ [21] has been introduced to augment the synchronization power of object types in the wait-free hierarchy. A failure detector of the class Ω^z outputs at each process a set of at most z process identities such that, after some time, the same set including the identity of at least one correct process is output at all correct processes. Clearly, Ω^1 is Ω. Moreover, $\Omega^z \subseteq \Omega^{z+1}$.

Motivation and results The paper first extends the family $(\phi^y)_{0 \leq y \leq n}$, by considering its eventual counterpart, namely the family $(\diamond\phi^y)_{0 \leq y \leq n}$. $\diamond\phi^y$ is a weakening of ϕ^y in the sense it allows the properties defining ϕ^y to be satisfied only after some finite time. So, while the families $(\mathcal{S}_x)_{1 \leq y \leq n}$ and $(\phi^y)_{0 \leq y \leq n}$ are characterized by a "perpetual" property (i.e., a property that has to be satisfied from the very begining), the families $(\diamond\mathcal{S}_x)_{1 \leq y \leq n}$, $(\Omega^z)_{1 \leq z \leq n}$ and $(\diamond\phi^y)_{0 \leq y \leq n}$ are characterized by an eventual property.

It appears that, when we are interested in solving set agremeent problems, we are provided with three families of failure detectors: $(\diamond\mathcal{S}_x)_{1 \leq x \leq n}$, $(\diamond\phi^y)_{0 \leq y < n}$ and $(\Omega^z)_{1 \leq z \leq n}$. Whatever the problems these failure detector classes help solving, important questions are the following: *Which among these classes are equivalent? Which are not? Is it possible to combine some of them to obtain stronger failure detector classes? If the answer is "yes", which ones and which failure detector class do they produce? If the answer is "no", why? Etc.* This is the type of questions addressed in this paper that characterizes relationships linking each pair of failure detector classes. More precisely, the issues and results of the paper are the following. The notation $A + B \rightsquigarrow C$ means that, given as inputs a failure detector of the class A and a failure detector of the class B, there is an algorithm that constructs a failure detector of the class C. The notation $A + B \not\rightsquigarrow C$ means that there is no such transformation algorithm. The notations $A \rightsquigarrow C$ and $A \not\rightsquigarrow C$ have the same meaning considering a single failure detector class as input.

- Reducibility, Irreducibility and Minimality.
  - Relations linking $\phi^y/\diamond\phi^y$ and $\mathcal{S}_x/\diamond\mathcal{S}_x$:
    - Let $1 \leq x \leq t+1$ and $1 \leq y \leq t$. $\mathcal{S}_x \not\rightsquigarrow \diamond\phi^y$. (Theorem 8.)
    - Let $0 \leq y < t$ and $1 < x \leq t+1$. $\phi^y \not\rightsquigarrow \diamond\mathcal{S}_x$. (Theorem 9.)
  - Relations linking $\phi^y/\diamond\phi^y$ and Ω^z:
    - $\diamond\phi^y \rightsquigarrow \Omega^z$ Iff $y + z > t$. (Corollary 6.)
    - Let $1 \leq z \leq t+1$ and $1 \leq y \leq t$. $\Omega^z \not\rightsquigarrow \diamond\phi^y$. (Theorem 10.)
  - Relations linking $\diamond\mathcal{S}_x$ and Ω^z:
    - $\diamond\mathcal{S}_x \rightsquigarrow \Omega^z$ Iff $x + z > t + 1$. (Corollary 7.)
    - Let $1 < x, z \leq t$. $\forall z : \Omega^z \not\rightsquigarrow \diamond\mathcal{S}_x$. (Theorem 11.)

  All these relations are depicted in Figure 1 where the bold arrows mean reducibility, and the dotted arrows mean irreducibility. The class $\mathcal{S}_x$ is the subclass of $\diamond\mathcal{S}_x$ where the accuracy is perpetual (namely, there is a correct process that is not suspected by x processes from the very beginning). $\mathcal{P}$ is the class of *perfect* failure detectors [3] (the ones that never do a mistake). The column at the right of the figure concerns k-set agreement: all the failure detector classes in the zth line allow solving z-set agreement. It is important to notice that (1) $\diamond\mathcal{S}_{t-z+2}$ and $\diamond\phi^{t-z+1}$ cannot be compared, (2) both are stronger than Ω^z. Moreover, given a line (say z) of the figure, Ω^z is the weakest failure detector class of that line that allows solving k-set agreement.

- Additivity. This paper poses (for the first time to our knowledge) the question of adding failure detectors of distinct classes. This is an important issue as "additivity" is a crucial concept as soon modularity and scalability of distributed systems are concerned.

  As an example, assuming $t > 1$, let us consider the class $\diamond\mathcal{S}_t$ that allows solving 2-set agreement (but not consensus), and the class $\diamond\phi^1$ that allows solving t-set agreement (but not $(t-1)$-set agreement).

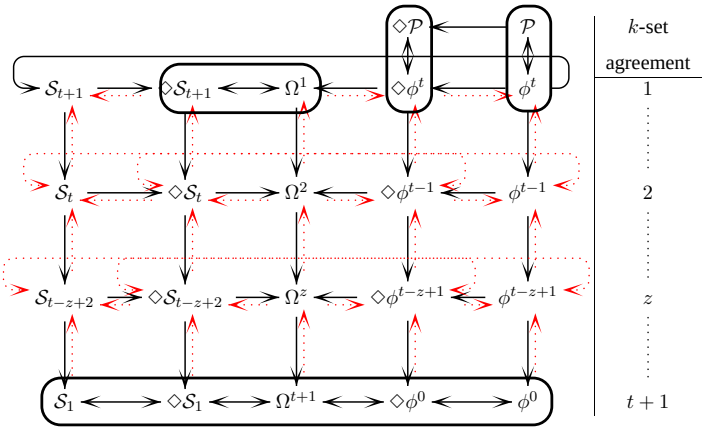


Figure 1: Grid of failure detector classes

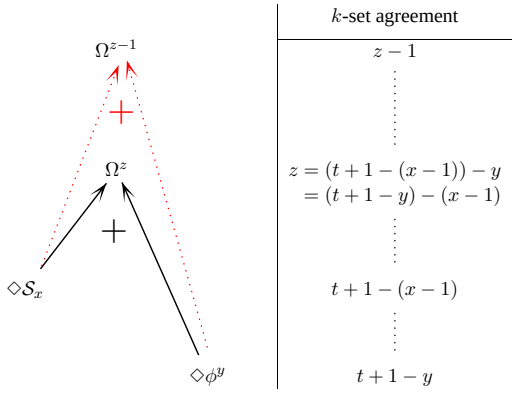


Figure 2: Additivity of $\diamond\mathcal{S}_x$ and $\diamond\phi^y$

What about $\diamond\mathcal{S}_t + \diamond\phi^1$? Is it possible to add them? If the answer is "yes", which type of information on failures is provided by their combination? The paper shows that $\diamond\mathcal{S}_t + \diamond\phi^{t-1}$ allows solving the consensus problem. More generally, with respect to the grid described in the previous figure, the paper characterizes which classes can be added and which cannot. More explicitly, it shows the following result (see also Figure 2): $\boxed{\diamond\mathcal{S}_x + \diamond\phi^y \rightsquigarrow \Omega^z \Leftrightarrow x+y+z > t+1.}$ To that end, the paper presents a construction algorithm (sufficiency part, Figures 5 and 6), and an impossibility proof (necessity part, Theorem 7).

Intuitively, this shows that $\diamond\mathcal{S}_x$ and $\diamond\phi^y$ provide different seeds to build Ω^z. To see the gain provided by such an addition, let us analyze it as follows:
- As $\diamond\mathcal{S}_x \rightsquigarrow \Omega^{t-x+2}$, the previous addition shows that adding ϕ^y allows strengthening Ω^{t-x+2} to obtain Ω^z with $z = (t-x+2)-y$.
- Similarly, as $\diamond\phi^y \rightsquigarrow \Omega^{t-y+1}$, the previous addition shows that adding $\diamond\mathcal{S}_x$ allows strengthening Ω^{t-y+1} to Ω^z with $z = (t-y+1)-(x-1)$.

- Asynchronous Ω^k-based k-set agreement. This paper proposes such an algorithm. (To our knowledge, no previous work has addressed the design of Ω^z-based k-set agreement algorithms.) The proposed algorithm (Figure 3) is very simple. Moreover, the paper establishes that $t < n/2$ and $z \leq k$ are two tight bounds of the k-set agreement problem, when considering the $(\Omega^z)_{1\leq z\leq n}$ family of failure detectors in an asynchronous message-passing system (Theorem 4). Consequently, among all the classes described in Figure 1, Ω^k is the weakest class for solving asynchronous k-set agreement (hence, the algorithm is optimal in that respect). This constitutes a step towards the characterization of the weakest failure detector class that allows solving the k-set agreement problem.

From a methodology point of view, the paper uses as much as possible reduction algorithms (striving not to reinvent the wheel). The paper presents also two more transformations for particular cases. These transformations (given in appendix) are simpler and more efficient than the general transformation building a failure detector of the class Ω^z from failure detectors of the classes $\diamond\mathcal{S}_x$ and $\diamond\phi^y$. The first (Figure 8) transforms Φ^y into Ω^z for $y+z > t$. The second (Figure 9) presents an addition algorithm of $\diamond\mathcal{S}_x$ with $\diamond\phi^y$ that provides $\diamond\mathcal{S}$ when $x+y > t$.

Roadmap The paper is made up of 5 sections plus an appendix. Section 2 describes the asynchronous computing model and the classes of failure detectors we are interested in. Section 3 presents the asynchronous Ω^k-based k-set agreement algorithm. Then, Section 4 presents an algorithm that builds a failure detector of the class Ω^z from a pair of underlying failure detectors, one of the class $\diamond\phi^y$, the other of the class $\diamond\mathcal{S}$.

Section 5 shows that $x+y+z > t+1$ is a necessary requirement for the previous construction, and establishes the irreducibility relations depicted by the grid of Figure 1.

2 Computation Model

2.1 Asynchronous System with Process Crash Failures

We consider a system consisting of a finite set Π of $n \geq 2$ processes, namely, $\Pi = \{p_1, p_2, \ldots, p_n\}$. When it is not ambiguous we also use Π to denote the set of the identities $1, \ldots, n$ of the processes. A process can fail by *crashing*, i.e., by prematurely halting. It behaves correctly (i.e., according to its specification) until it (possibly) crashes. By definition, a process is *correct* in a run if it does not crash in that run; otherwise it is faulty. As previously indicated, t denotes the maximum number of processes that can crash in a run ($1 \leq t < n$). The identity of the process p_i is i, and each process knows all the identities.

Processes communicate and synchronize by sending and receiving messages through channels. Every pair of processes is connected by a channel. Channels are assumed to be reliable: they do not create, alter or lose messages. In particular, if p_i sends a message to p_j, then eventually p_j receives that message unless it fails. There is no assumption about the relative speed of processes or message transfer delays (let us observe that channels are not required to be FIFO).

Broadcast(m) is an abbreviation for "**foreach** $p_j \in \Pi$ **do** *send*(m) *to* p_j **enddo**". Moreover, we assume (without loss of generality) that the communication system provides the processes with a *reliable broadcast* abstraction [10]. Such an abstraction is made up of two primitives *Broadcast()* and *Deliver()* that allow a process to broadcast and deliver messages (we say accordingly that a message is R_broadcast or R_delivered by a process) and satisfy the following properties:

- Validity: If a process R_delivers m, then some process has R_broadcast m. (No spurious messages.)
- Integrity: A process R_delivers a message m at most once. (No duplication.)
- Termination: If a correct process R_broadcasts or R_delivers a message m, then all the correct processes R_delivers m. (No message R_broadcast or R_delivered by a correct process is missed by a correct process.)

As we can see, the messages sent (resp., R_broadcast) by a process are not necessarily received (resp., R_delivered) in their sending order. Moreover, different processes can R_deliver messages in different order. There is no assumption on message transfer delays. The communication system is consequently reliable and asynchronous.

2.2 Failure Detector Classes

The definition of the families of failure detector classes $(\mathcal{S}_x)_{1\leq x\leq n}$, $(\diamond\mathcal{S}_x)_{1\leq x\leq n}$, $(\Phi^y)_{0\leq y<n}$, $(\phi^y)_{0\leq y<n}$ and $(\Omega^z)_{1\leq z\leq n}$ have been sketched in the introduction. This section provides more complete definitions.

The classes $(\mathcal{S}_x)_{1\leq x\leq n}$ and $(\diamond\mathcal{S}_x)_{1\leq x\leq n}$ A failure detector of the class $\mathcal{S}_x$ or $\diamond\mathcal{S}_x$ consists of a set of modules, each one attached to a process: the module attached to p_i maintains a set (named $suspected_i$) of processes it currently suspects to have crashed. As in other papers devoted to failure detectors, we say "process p_i suspects process p_j at some time τ", if $p_j \in suspected_i$ at that time. Moreover, (by definition) a crashed process suspects no process.

The failure detector $\diamond\mathcal{S}_x$ class generalizes the class $\diamond\mathcal{S}$ defined in [3] (we have $\diamond\mathcal{S}_n = \diamond\mathcal{S}$). A failure detector belongs to the class $\diamond\mathcal{S}_x$ if it satisfies the following properties:

- Strong Completeness: Eventually, every process that crashes is permanently suspected by every correct process.
- Limited Scope Eventual Weak Accuracy: There is a time after which there is a set Q of x processes such that Q contains a correct process and that process is never suspected by the processes of Q.

Similarly, the class $\mathcal{S}_x$ generalizes the class $\mathcal{S}$ [3] (and we have $\mathcal{S}_n = \mathcal{S}$). A failure detector of the class $\mathcal{S}_x$ satisfies the previous strong completeness property, plus the following accuracy property:

- Limited Scope Perpetual Weak Accuracy: there is a set Q of x processes such that (from the very beginning) Q contains a correct process and that process is never suspected by the processes of Q.

It is easy to see that $\mathcal{S}_{x+1} \subseteq \mathcal{S}_x$, $\diamond\mathcal{S}_{x+1} \subseteq \diamond\mathcal{S}_x$, and $\mathcal{S}_x \subseteq \diamond\mathcal{S}_x$. It is also easy to see that the failure detectors of the classes $\mathcal{S}_1$ and $\diamond\mathcal{S}_1$ provide no information on failures. It has been shown in [11] that $\diamond\mathcal{S}_x$ is the weakest failure detector class of the family $(\diamond\mathcal{S}_x)_{1\leq x\leq n}$ that allows solving k-set agreement for $k = t - x + 2$, in asynchronous message-passing systems with a majority of correct processes ($t < n/2$).

The classes $(\Omega^z)_{1\leq z\leq n}$ This family of failure detectors has been introduced in [21]. A failure detector of the class Ω^z maintains at each process p_i a set of processes of size at most z (denoted $trusted_i$) that satisfies the following property:

- Eventual Multiple Leadership: there is a time after which the sets $trusted_i$ of the correct processes contains forever the same set of processes and at least one process of this set is correct.

The family $(\Omega^z)_{1\leq z\leq n}$ generalizes the class of failure detectors Ω defined in [2] (we have $\Omega^1 = \Omega$).

Recently, another generalization of Ω has been studied in [7] that considers Ω_S, where S is a predefined subset of the processes of the system. Ω_S requires that all the correct processes of S eventually agree on the same correct leader (it is not required that their eventual common leader belongs to S). Let X be the set of all the pairs of processes. It is shown in [7] that, given all the Ω_x, $x \in X$, it is possible to build Ω.

The classes $(\Phi^y)_{0\leq y<n}$ and $(\phi^y)_{0\leq y<n}$ These failure detector classes have been introduced in [16] to solve set agreement problems in combination with conditions [15]. Differently from the previous classes of failure detectors that provides each process p_i with a set ($suspected_i$ or $trusted_i$) that p_i can only read, a failure detector of a class ϕ^y (or Φ^y) provides the processes with a primitive QUERY(X), where X is a set of process identities supplied by the invoking process. Such a primitive allows a process p_i to query about the crash of a region X of the system. More precisely, a failure detector of the class ϕ^y is defined by the following properties (remind that t is an upper bound on the number of process crashes):

- Triviality property. If $|S| \leq t - y$, then QUERY$_y(X)$ returns *true*. If $|S| > t$, then QUERY(X) returns *false*.
- Safety property. If $t - y < |X| \leq t$, then if at least one process in X has not crashed when QUERY(X) is invoked, the invocation returns *false*.
- Liveness property. Let X be such that $t - y < |X| \leq t$. Let τ be a time such that, at time τ, all the processes in X have crashed. Moreover, let us assume that after τ there is an infinite sequence of invocations of QUERY(X). Then, for some time $\tau' \geq \tau$, all the invocations of QUERY(X) return *true*.

The triviality property provides the invoking process with a trivial output when the set X is too small or too big. The safety property states that if the output is *true*, then all the processes in X have crashed. The liveness property states that QUERY(X) eventually outputs *true* when all the processes in X have crashed. It is shown in [16] that (1) $\phi^{y+1} \subseteq \phi^y$, and (2) ϕ^t and the class $\mathcal{P}$ of perfect failure detectors are equivalent in any system where at most t processes can crash. Moreover, it is easy to see that ϕ^0 provides no information on failures.

The class Φ^y is a subclass of ϕ^y. It additionally requires that for any pair of queries QUERY($X1$) and QUERY($X2$), we have $X1 \subseteq X2$ or $X2 \subseteq X1$. Hence, we have $\Phi^y \subset \phi^y$. It has been shown in [16] that, within the family $(\Phi^y)_{0\leq y\leq t}$ of failure detector classes, Φ^y is the weakest for solving k-set agreement for $k = t - y + 1$, in asynchronous shared memory systems.

The classes $(\diamond\phi^y)_{0\leq y<n}$ The failure detector class $\diamond\phi^y$ is the "eventual" counterpart of the class ϕ^y. More precisely, a failure detector of the class $\diamond\phi^y$ is defined by the following properties (remind that t is an upper bound on the number of process crashes):

- Triviality property. If $|S| \leq t - y$, then QUERY$_y(X)$ returns *true*. If $|S| > t$, then QUERY(X) returns *false*.
- Eventual Safety property. Let X be such that $t - y < |X| \leq t$. Suppose that at least one correct process belongs to X. Moreover, let us assume that there is an infinite sequence of invocation of QUERY(X). Then, it exists some time τ from which all the invocations of QUERY(X) return *false*.
- Liveness property. Let X be such that $t - y < |X| \leq t$. Let τ be a time such that, at time τ, all the processes in X have crashed. Moreover, let us assume that after τ there is an infinite sequence of invocations of QUERY(X). Then, for some time $\tau' \geq \tau$, all the invocations of QUERY(X) return *true*.

As for the classes $(\phi^y)_{0\leq y\leq t}$, it follows from these properties that (1) $\diamond\phi^{y+1} \subseteq \diamond\phi^y$, and (2) $\diamond\phi^t$ and the class $\diamond\mathcal{P}$ are equivalent in any system where at most t processes can crash.

2.3 Notation

Let $\mathcal{F}$ and $\mathcal{G}$ be any two classes among the previous classes of failure detectors. The notation $\mathcal{AS}_{n,t}[\mathcal{F}]$ is used to represent a message-passing asynchronous system made up of n processes, where up to t may crash, equipped with a failure detector of the class $\mathcal{F}$. Similarly, $\mathcal{AS}_{n,t}[\mathcal{F},\mathcal{G}]$ denotes a system equipped with a failure detector of the class $\mathcal{F}$ and a failure detector of the class $\mathcal{G}$. Finally, $\mathcal{AS}_{n,t}[\emptyset]$ denotes a "pure" asynchronous message-passing system (i.e., without additional equipment).

3 Using Ω^k to Solve k-Set Agreement

This section presents an Ω^k-based k-set agreement algorithm, and lower bounds when solving k-set agreement with failure detector classes of the family $(\Omega^k)_{1\leq z\leq n}$. These lower bounds are $t < n/2$ and $z \leq k$. Interestingly, the proof of these bounds is based on a reduction to a $\diamond\mathcal{S}_x$-based k-set agreement algorithm and a corresponding lower bound [11].

3.1 A k-Set Agreement Algorithm

The algorithm, described in Figure 3, is a simple adaptation of an Ω-based consensus algorithm described in [9] (which is in turn inspired from a $\diamond\mathcal{S}$-based consensus algorithm described in [18]); it assumes $t < n/2$. A process p_i invokes k-SET_AGREEMENT(v_i), where v_i is the value it proposes. If it does not crash, it terminates when it executes the statement *return*(v), where v is then the value it decides.

The function k-SET_AGREEMENT(v_i) is made up of two tasks. The task $T2$ is used to disseminate a decided value and prevent deadlock: due to the reliable broadcast, as soon as a process decides, all the correct processes decide. In the main task $T1$, the processes proceed in consecutive asynchronous rounds, each round being made up of two phases, each including a communication step. When considering p_i, the local variable est_i is the local estimate of the decision value; r_i is its current round number.

During the first phase of round r, p_i first reads $trusted_i$ (the set provided by its underlying failure detector module of the class Ω^z), stores its value in L_i, and sends a PHASE1(r_i, L_i, est_i) message to all the processes. Then, p_i waits until it has received such round r messages from $n-t$ processes (i.e., from at least a majority) and it has either received such a message from a process of its L_i set or the set $trusted_i$ has changed. Then, if a majority of processes have the same leader set L, and p_i has received an estimate value v_L from a process in this set L, it keeps v_L in aux_i, otherwise it sets aux_i to $\perp$. Let us notice that we can conclude from the previous statements (see the proof) that, at the end of the first phase of each round, the set of the aux_i local variables contains at most $|L_i| = k$ distinct values different from $\perp$.

The second phase of a round aims at allowing the processes to decide, while ensuring that no more than k different values are decided, whatever the round during which a process decides. To that end, each process

p_i broadcasts a PHASE2(r_i, aux_i) message to all the processes, and then waits until it has received such messages from $n - t$ processes. If it receives a non-$\perp$ value v, it adopts v as its new estimate (if there are several such values, it takes one arbitrarily). Moreover, if none of the values it has received is $\perp$, it decides the estimate value v it has just adopted; this is done by broadcasting v in a reliable way, and then returning that value (in task $T2$).

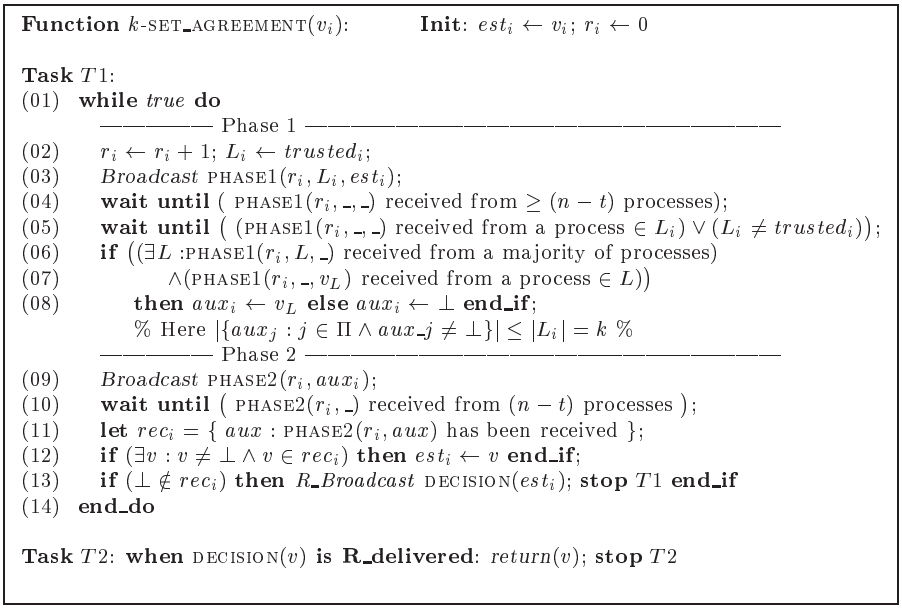


Figure 3: Ω^k-based k-set agreement algorithm (code for p_i)

3.2 Short Discussion

The notion of *perfection, oracle-efficiency* and *zero-degradation* used here are straightforward generalizations of the same notions introduced in [6, 9] in the context of failure detector-based consensus algorithms.

Let a failure detector of the class Ω^k be *perfect* if, from the very beginning, it delivers to the processes the same set of at most k processes including at least one correct process. A set agreement algorithm is *oracle-efficient* if it terminates in two communication steps (a single round) when its underlying failure detector is perfect and there is no crash. It is easy to see that the previous algorithm is oracle-efficient. This algorithm satisfies an even stronger property, namely, it is zero-degrading. A set agreement algorithm is *zero-degrading* if it terminates in two steps when its underlying failure detector is perfect and there are only initial crashes (a crash is *initial* if the corresponding process crashes before the algorithm starts). The reader can easily checks that the proposed algorithm is zero-degrading. Zero-degradation is particularly important when a set agreement algorithm is used repeatedly: it means that future executions do not suffer from past process failures as soon as the failure detector behaves perfectly.

3.3 Proof of the Algorithm

The proof is similar to the proof of the Ω-based consensus algorithm described in [9]. It assumes $t < n/2$ and $z \leq k$ (see Theorem 4).

Lemma 1 *No correct process blocks forever in a round.*

Proof Let p_i be a correct process. We have to show, whatever the round number r, that p_i cannot be blocked forever in the **wait** statements (lines 04, 05 and 10) of round r. This follows from (1) the fact that t being the maximum number of faulty processes, (2) the termination and integrity properties of the reliable

broadcast primitive, as well as (3) the fact that the leader set eventually permanently contains a correct process. In more detail, we have the following.

If a process decides, then by the termination property of the reliable broadcast of the corresponding DECISION() message, every correct process decides, and consequently no correct process can block forever in a round. Assume by contradiction that no process decides. Let r be the smallest round in which some correct process p_i blocks forever. So p_i blocks at line 04, 05 or 10. Consider the case of line 04. Since no correct process blocks in a round $r' < r$ and no correct process decides, all correct processes broadcast a PHASE1$(r, _, _)$ message. As the maximum number of faulty processes is t, it follows from the integrity and termination of the broadcast primitive that p_i eventually delivers $n - t$ such messages. Consequently, p_i cannot block at line 04. The fact that p_i cannot block forever at line 05 follows directly from the fact that its local set $trusted_i$ eventually permanently contains the identity of a correct process and the fact that all the correct processes broadcast a PHASE1$(r_, _)$ message. Consider line 10: as we have just shown that no correct process blocks forever in phase 1 of round r, it follows that all correct processes broadcast a PHASE2$(r, _, _)$ message. Consequently (as in line 04), p_i does not block forever at line 10. $\Box_{Lemma\ 1}$

Assuming p_i completes line 08 during round r, let $aux_i[r]$ be the value of aux_i after it has been updated by p_i at line 08. Moreover, let $AUX[r] = \{aux_i[r] \mid p_i$ completes phase 1 of $r\ \}$.

Lemma 2 $\forall r : |\{v : v \in AUX[r] \wedge v \neq \bot\}| \leq k$.

Proof Let p_i be a process that completes phase 1 of round r. Let us observe that p_i sets aux_i to a value $v \neq \bot$ only if it sees that a majority of processes have the same leader set L (lines 06-08). Moreover, v is a value proposed by a process that belongs to L. Let us notice that there is at most one set that is considered leader set by a majority of processes. Consequently, all the values $aux_i \neq \bot$ at the end of the round r are estimate values of processes belonging to the same set L. Since this set is of size k, it follows that $|\{aux_i[r] : aux_i[r] \neq \bot \wedge p_i$ completes phase 1 of round $r\}| \leq k$. $\Box_{Lemma\ 2}$

Lemma 3 *Suppose that no process decides.* $\exists r : \bot \notin AUX[r]$.

Proof It follows from the eventual multiple leadership of the class Ω^k that there is a time τ after which all the processes have permanently the same leader set L and this set contains a correct process. Let r be a round that starts after that time (i.e., the first process, say p_i, that executes $r_i \leftarrow r$ does so at time $\tau' > \tau$). As no correct process blocks in the round r (Lemma 1), each correct process broadcasts PHASE1$(r, _, _)$, from which it follows that the condition of the **if** statement of line 06-07 is satisfied for all the processes that complete phase 1 of round r. Consequently, no process p_i sets its aux_i variable to $\bot$. $\Box_{Lemma\ 2}$

Theorem 1 [Validity] *Any decided value is a proposed value.*

Proof The special value $\bot$ cannot be decided (lines 12-13). Moreover, it follows from the integrity and validity of the broadcast primitive that the aux_i and est_i variables can only contain proposed values or $\bot$. $\Box_{Theorem\ 1}$

Theorem 2 [Agreement] *At most k distinct values are decided.*

Proof If no process decides, the theorem is trivially true. So, let us assume that a process decides and let r be the smallest round during which some process decides ("decide v during r" means "during r, execute line 13 with $\bot \notin rec_i \wedge est_i = v$"). We first show that there is a set V of values, $|V| \leq k$, such that any process that decides during r decides a value from V. We then show any value decided during a subsequent round belongs to V.

Let $V = \{v : v \in AUX[r] \wedge v \neq \bot\}$. us first notice that $|V| \leq k$ (Lemma 2). Let p_i be a process that decides during round r. Let $rec_i[r]$ be the value of the set rec_i computed at line 11 of round r. Let us observe that $rec_i[r] \subseteq AUX[r]$ (lines 10-11). Since p_i decides a value $v \neq \bot$ in $rec_i[r]$, we have $v \in V$.

Assuming that some process p_i decides a value $v \in V$ during round r, we now prove that the estimate est_j of any process p_j that progresses to $r+1$ belongs to V. As there are at least $n-t$ PHASE2$(r, _)$ messages

carrying a value $aux \neq \perp$ (these are the messages that allowed p_i to decide during round r) and $n - t > n/2$, it follows from the integrity and validity properties of the broadcast primitive that p_j has received at least one of these PHASE2 messages. Consequently, when p_j executes line 12, it updates its estimate to a value $aux \neq \perp$. Hence, from definition of set V we have $est_j \in V$. It follows that estimate est_j of all the processes p_j that start the round $r + 1$ belong to V. $\square_{Theorem\ 2}$

Theorem 3 [Termination] *Every correct process eventually decides.*

Proof The proof is by contradiction. Assume that no correct process decides. By Lemma 1, the correct processes progress from round to round. Hence, due to Lemma 3, there is a round r such that $\perp \notin AUX[r]$. Consequently, any PHASE2(r, aux) that is broadcast is such that $aux \neq \perp$. Due to the integrity and termination properties of the broadcast primitive, we have $\perp \notin rec_i$ for any process p_i executing the second phase of round r. We can then conclude (line 13) that the correct processes decide: a contradiction. $\square_{Theorem\ 3}$

3.4 A Lower Bound

Considering an asynchronous message-passing system equipped with a failure detector of the class Ω^z, $1 \leq z \leq n$, this section establishes that $t < n/2$ and $z \leq k$ are necessary and sufficient conditions for solving the k-set agreement problem. As already noticed, this result is obtained by a reduction to the problem of the weakest failure detector in the family $(\diamond\mathcal{S}_x)_{1 \leq x \leq n}$ that allows solving k-set agreement.

Theorem 4 *The k-set agreement problem is solvable in $\mathcal{AS}_{n,t}[\Omega^z]$ if and only if $t < n/2$ and $z \leq k$.*

Proof [$\Rightarrow$ part] The proof is by contradiction. let us assume that there is an algorithm $\mathcal{A}$ that solves the k-set agreement problem in $\mathcal{AS}_{n,t}[\Omega^z]$ such that $t \geq n/2$ or $z > k$. Due to Theorem 7, there is an algorithm $\mathcal{T}$ that builds a failure detector of the class Ω^z in $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_{t-z+2}]$. Moreover, there are such transformation algorithms (e.g., the one presented in Section 4 with $y = 0$) that are independent of the value of t (i.e., $t < n$). Combining such a transformation $\mathcal{T}$ and the algorithm $\mathcal{A}$, we obtain an algorithm that solves the k-set agreement problem in $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_{t-z+2}]$. It then follows from the lower bound established by Herlihy and Penso [11] for solving the k-set agreement problem in $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_{t-z+2}]$ that $t < \min(n/2, (t - z + 2) + k - 1)$, from which we conclude $t < n/2$ and $z \leq k$: a contradiction.
[$\Leftarrow$ part] This part follows directly from the very existence of the Ω^k-based k-set agreement algorithm described in Section 3.1 and proved in Section 3.3. $\square_{Theorem\ 4}$

4 Additivity of the Failure Detector Classes $\diamond\mathcal{S}_x$ and $\diamond\phi^y$

This section presents an algorithm that, given as input any pair of failure detectors of the classes $\diamond\mathcal{S}_x$ and $\diamond\phi^y$, constructs a failure detector of the class Ω^z, provided that $x + y + z > t + 1$. (It is proved in Section 5.1 that this is a necessary requirement for such a construction, thereby showing that the algorithm is optimal.)

The algorithm is made up of two components that we call *wheels* because each "turns" like a gear-wheel until they become synchronized and stop turning. The wheel that is the first to eventually stop is the one whose progress depends on the the underlying $\diamond\mathcal{S}_x$ failure detector ("lower" wheel). When it stops, it provides a property that allows the second wheel in turn to eventually stop ("upper" wheel). As we will see, the wheel metaphor comes from the fact that each component is made up of main tasks that "turn", each scanning a sequence until some property becomes satisfied.

Let us remind that $1 \leq x \leq n$. Moreover, as the class $\diamond\phi^t$ is equivalent to the class of eventual perfect failure detectors we consider only the cases $0 \leq y \leq t$, from which we conclude $t - y + 1 > 0$. Finally, as $z \geq t + 2 - (x + y)$ is a necessary requirement and Ω^1 is the strongest class in the family $(\Omega^z)_{1 \leq z \leq n}$, the only interesting cases for the pair (x, y) are when $t + 2 - (x + y) \geq 1$. Hence, in the following we consider that $t - y + 1 > 0$, $z = t + 2 - (x + y)$ and $t + 2 - (x + y) > 0$.

4.1 The Lower Wheel Component

4.1.1 Description

The aim of this component is to provide each process p_i with a local variable $repr_i$ intended to contain a process identity and such that the following property becomes eventually satisfied: there is a set X of x processes that either have crashed, or the variables $repr_i$ of the processes of X that have not crashed, contains the identity ℓx of one of them that is a correct process. This process is their common representative (leader). The variable $repr_i$ of a process p_i that does not belong to X has to be equal to the identity i of p_i.

To attain this goal the processes use their local sets $suspected_i$ that collectively satisfy the completeness and limited scope eventual accuracy properties defining the class $\diamond\mathcal{S}_x$. Let $\mathcal{X}$ be the finite set of all the sets of x processes that can be built from the set $\Pi = \{p1, \ldots, p_n\}$. Let nb_x denote the number of combinations of x elements in a set of n elements. $\mathcal{X}$ has nb_x elements. Let us organize $\mathcal{X}$ as a sequence, and let $\mathcal{X}[k]$ be its kth element, $1 \leq k \leq nb\_x$. Within $\mathcal{X}[k]$, let us arrange the x processes it is made up of in some predefined (arbitrary) order: $\ell_1^k, \ldots, \ell_x^k$. This means that the infinite sequence $\mathcal{X}[1], \mathcal{X}[2], \ldots, \mathcal{X}[nb\_x], \mathcal{X}[1], \mathcal{X}[2], \ldots, \mathcal{X}[nb\_x], \mathcal{X}[1], \ldots$ gives rise to an infinite sequence of process identities, namely, $\ell_1^1, \ldots, \ell_x^1, \ell_1^2, \ldots, \ell_x^2, \ell_1^3, \ldots$ (see Figure 4). This sequence is assumed to be initially known by all the processes in order they can scan it in the same order.

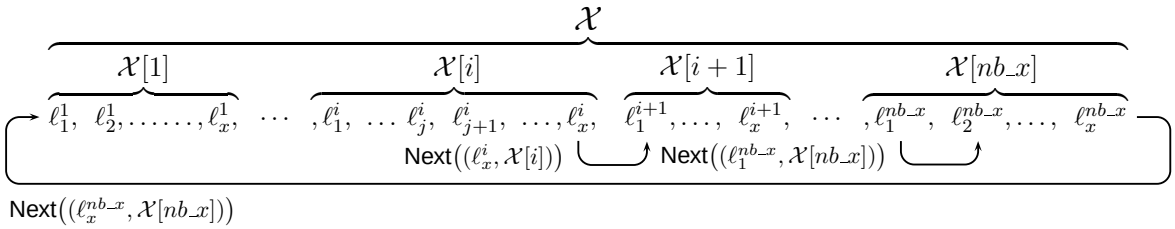


Figure 4: The Next() function on the logical ring (ℓ, X)

In addition to its output $repr_i$, each process p_i manages a local set X_i and a local variable ℓx_i. It starts with X_i initialized to $\mathcal{X}[1]$, and ℓx_i initialized to ℓ_1^1 (the first process of $\mathcal{X}[1]$). Then, it uses the function Next$(-,-)$ defined as follows to progress along the infinite sequence of process identities. Next$(\ell_y^k, \mathcal{X}[k])$ outputs the pair $(\ell_{y+1}^k, \mathcal{X}[k])$ if $y < x$ and the pair $(\ell_1^{k+1}, \mathcal{X}[k+1])$ if $y = x$ (with $k+1$ being replaced by 1 when $k = nb\_x$).

```
Init: X_i ← X[1]; ℓx_i ← ℓ_1^1; repr_i ← i

Task T1:
    repeat forever
        if (i ∈ X_i) then repr_i ← ℓx_i else repr_i ← i end_if;
        if ((i ∈ X_i) ∧ (ℓx_i ∈ suspected_i)) then R_Broadcast X_MOVE(ℓx_i, X_i) end_if
    end_repeat

Task T2: when X_MOVE(ℓx_i, X_i) is R_delivered: (ℓx_i, X_i) ← Next(ℓx_i, X_i)
```

Figure 5: From $\diamond\phi^y + \diamond\mathcal{S}_x$ to Ω^z: lower wheel component (code for p_i)

The behavior of the lower wheel component of a process p_i is described in Figure 5. It is made up of two simple tasks. The processes scan the infinite sequence of sets generated from $\mathcal{X}$ until they stabilize. X_i represents the set of x processes that are currently in charge of extracting a common representative ℓx_i from this set. To do it, each process p_i that belongs to X_i uses its set $suspected_i$ provided by the underlying failure detector of the class $\diamond\mathcal{S}$. If the processes of X_i succeed in not suspecting one of them -whose identity is kept by p_i in ℓx_i-, they stop sending X_MOVE() messages. Differently, if a process p_j of the set X_i suspects its current "leader" ℓx_j, it uses the reliable broadcast primitive to send the message X_MOVE$(\ell x_j, X_i)$ indicating

that, from its point of view, ℓx_j cannot be their common representative. A process p_j delivers a message X_MOVE$(\ell x, X)$ only when $\ell x = \ell x_i$ and $X_i = X$; it then proceeds to the next process identity (according to the infinite sequence), and possibly to the next candidate set $\mathcal{X}[k+1]$ if $X_i = X = \mathcal{X}[k]$ and $\ell x = \ell x_i$ is the last process of $\mathcal{X}[k]$.

Let us finally consider the case where the processes progress until they consider a set X such that the x processes that constitute X have crashed. It is easy to see that each no-crashed process p_i continues looping inside task $T1$ without sending messages, and is such that $repr_i = i$.

4.1.2 Proof of the lower wheel component

The proof considers an arbitrary run of the algorithm described in Figure 5. C denotes the set of processes that are correct in that run. Moreover, var_i^τ denotes the value of the local variable var_i at time τ.

Lemma 4 *$\forall i \in C$, there are a pair (λ_i, σ_i) and a time τ_i such that $\forall \tau \geq \tau_i : (\ell x_i^\tau, X_i^\tau) = (\lambda_i, \sigma_i)$.*

Proof We claim (*Claim C1*) that there is a pair (ℓ, X) such that the number of X_MOVE(ℓ, X) messages that are sent is finite. Let us assume (by way of contradiction) that there is no pair (λ_i, σ_i) such that after some time $(\ell x_i, X_i) = (\lambda_i, \sigma_i)$ remains true forever. As the pairs $(\ell x, X)$ are arranged in a logical ring (see Figure 4), it follows from the way p_i updates its local pair $(\ell x_i, X_i)$ that the sequence of the successive values of the local variables $(\ell x_i, X_i)$ is $(\ell_1^1, \mathcal{X}[1]), (\ell_1^2, \mathcal{X}[1]), \ldots, (\ell_x^{nb\_x}, \mathcal{X}[nb\_x]), (\ell_1^1, \mathcal{X}[1])$, etc. Consequently, $(\ell x_i, X_i)$ takes each values $(\ell_\alpha^\beta, \mathcal{X}[\beta]), 1 \leq \alpha \leq x, 1 \leq \beta \leq nb\_x$ infinitely often. In particular, p_i executes $(\ell x_i, X_i) \leftarrow \mathsf{Next}(\ell, X)$ infinitely often. But this contradicts the Claim *C1* that states that the number of X_MOVE(ℓ, X) messages that are sent is finite. It follows that there are a pair (λ_i, σ_i) and a time τ_i such that $\forall \tau \geq \tau_i : (\ell x_i^\tau, X_i^\tau) = (\lambda_i, \sigma_i)$.

Claim C1: There is a pair (ℓ, X) such that the number of X_MOVE(ℓ, X) messages that are sent is finite.
Proof of Claim C1. We consider two cases according to the number f of actual process crashes.

- Case 1: $f \geq x$. Let X be a set of x processes that are faulty and ℓ be the identity of an arbitrary process in X. As only processes that belongs to X can send X_MOVE(ℓ, X) messages, it follows from the fact all these processes eventually stop taking steps that the number of X_MOVE(ℓ, X) messages sent is finite.
- Case 2: $f < x$. Due to the limited scope eventual accuracy property of the class $\diamond\mathcal{S}_x$, there are a set $X \subseteq \Pi$ of size x and a correct process $p_\ell \in X$ such that, after some time τ, no process that belongs to the set X suspects p_ℓ. Since (1) only process that belongs to X can send X_MOVE(ℓ, X) messages, and, (2) a process $p_i \in X$ broadcasts a X_MOVE(ℓ, X) message only if $\ell \in suspected_i$, it follows that after time τ, no message X_MOVE(ℓ, X) can be broadcast, which implies that the number of such messages is finite. *End of the Proof of Claim C1.*

$\square_{Lemma\ 4}$

Corollary 1 *The protocol is quiescent (i.e., eventually all the processes stop sending* X_MOVE *messages).*

Proof Let us assume (for contradiction) that there is a correct process p_i that never stop sending X_MOVE messages. Due to Lemma 4, there is a time τ after which $(\ell x_i, X_i)$ remains permanently equal to the constant pair (λ_i, σ_i). Consequently, after time τ, p_i keeps on broadcasting X_MOVE(λ_i, σ_i). It follows then from the validity and termination properties of the reliable broadcast primitive that there is a time $\tau' > \tau$ at which p_i executes $(\ell x_i, X_i) \leftarrow \mathsf{Next}(\lambda_i, \sigma_i)$, contradicting Lemma 4. $\square_{Corollary\ 1}$

Lemma 5 *$\forall i, j \in C : (\lambda_i, \sigma_i) = (\lambda_j, \sigma_j)$. (In the following, (λ, σ) denotes that pair.)*

Proof Due to the properties of the reliable broadcast primitive, p_i and p_j deliver the same multiset of X_MOVE(ℓ, X) messages. Moreover, it follows from Corollary 1 that this multiset is finite. Due to the fact that p_i and p_j consume the messages according to the same ring order, and the fact that the common multiset of delivered messages is finite, it follows that $(\lambda_i, \sigma_i) = (\lambda_j, \sigma_j)$. $\square_{Lemma\ 5}$

Lemma 6 $(\sigma \cap C \neq \emptyset) \Rightarrow (\lambda \in C)$.

Proof Let us assume (by contradiction) that $\sigma \cap C \neq \emptyset$ and λ is the identity of a faulty process. Let p_i be a process that belongs to $\sigma \cap C$. Due to the strong completeness property of the class $\diamond\mathcal{S}_x$, it exist a time $\tau 1$ after which the local predicate $\lambda \in suspected_i$ remains permanently satisfied. Moreover, it follows from Lemmas 4 and 5 that, from some time τ_i, the predicate $(\ell x_i, X_i) = (\lambda, \sigma)$ remains permanently true. There is consequently a time $\tau \geq \max(\tau 1, \tau_i)$ at which p_i broadcasts a message X_MOVE(λ, σ). When p_i delivers this message, it executes $(\ell x_i, X_i) \leftarrow \mathsf{Next}(\lambda, \sigma)$, contradicting Lemma 4. $\square_{Lemma\ 6}$

Theorem 5 *The algorithm described in Figure 5 ensures the existence of a set X and a time τ such that $\forall \tau' \geq \tau$, the following holds:*

1. $|X| = x$,
2. $i \in \Pi - X \Rightarrow repr_i = i$,
3. $\forall i, j \in X \cap C : repr_i = repr_j = \rho \in C \cap X$.

Proof Let $\tau = \max\{\tau_i : i \in \Pi\}$ where τ_i is the time introduced in Lemma 4, and σ and λ be the set and the process identity defined in Lemma 5. Let us first observe that due to its definition (σ is a set X_i) we have $|\sigma| = x$ (1). Let p_i be a correct process. If $i \in \Pi - X$, then as the value of $repr_i$ does not change after time τ (Lemma 4 and Task $T1$), it follows that $repr_i = i$ is permanently true from time τ (2). Moreover, it directly follows from Lemma 5 and task $T1$ that all the correct processes p_j belonging to the set σ have permanently the same representative $repr_j = \lambda$ from time τ. Finally, λ is the identity of a correct process sue to Lemma 6 (3). Taking $X = \sigma$, $\tau = \max\{\tau_i : i \in \Pi\}$ and $\rho = \lambda$ completes the proof of the theorem. $\square_{Theorem\ 5}$

4.2 The Upper Wheel Component

4.2.1 Principles and description

The "upper wheel" component consists of four tasks $T3 - T6$ (Figure 6). Similarly to the lower wheel component, it uses a set, that we call $\mathcal{Y}$, including all the possible sets of $t - y + 1$ processes built from the n processes composing the system. Let nb_y denote the number of such distinct sets. Organizing $\mathcal{Y}$ as a sequence, let $\mathcal{Y}[k]$ be its kth element, and let us consider the infinite sequence $\mathcal{Y}[1], \mathcal{Y}[2], \ldots, \mathcal{Y}[nb\_y], \mathcal{Y}[1], \ldots$ Moreover, given any set $\mathcal{Y}[k]$ of this sequence, let us consider all its subsets of size $z = (t+2) - (x+y)$ (let nb_L denote the number of such subsets). Finally, let us order them (the order is arbitrary). Let $L_1^k, L_2^k, \ldots, L_{nb\_L}^k$ denote the sequence of all the sets of size $(t+2) - (x+y)$ generated from the set $\mathcal{Y}[k]$ (whose size is $t - y + 1$). As before, the infinite sequence $L_1^1, L_2^1, \ldots, L_{nb\_L}^1, L_1^2, \ldots, L_{nb\_L}^2, \ldots, L_1^{nb\_y}, \ldots, L_{nb\_L}^{nb\_y}, L_1^1, L_2^1, \ldots$ is initially known by each process. The function $\mathsf{Next}(L_r^k, \mathcal{Y}[k])$ is defined similarly to the previous $\mathsf{Next}()$ function. It outputs $(L_{r+1}^k, \mathcal{Y}[k])$ if $r < nb\_L$, and outputs $(L_1^{k+1}, \mathcal{Y}[k+1])$ if $r = nb\_L$ ($k+1$ being replaced by 1 when $k = nb\_y$).

Given these ingredients we can now describe the principles the additive transformation relies on. The aim is for p_i to return L_i as the value of the set $trusted_i$ it provides to the upper layer (remind that this set has to include at most z processes and eventually at least one correct process). Within the upper wheel component, the processes start from the set $\mathcal{Y}[1]$ and then scan the same infinite sequence of sets $\mathcal{Y}[1], \mathcal{Y}[2], \ldots, \mathcal{Y}[nb\_y], \mathcal{Y}[1], \ldots$ (tasks $T3$ and $T4$). When p_i is working with Y_i, it looks for one of its subset L_i (of size z) containing a correct process (when this occurs, that set defines the value of $trusted_i$). To check if its current set L_i contains a correct process, p_i sends INQUIRY() messages and waits until it has received at least one RESPONSE(id) message from a process of Y_i or all the processes of the set Y_i have crashed (task $T3$). When a process p_j sends back a response, it sends the last identity $repr_j$ currently computed by its underlying wheel (task $T5$). Let us consider two cases.

```
Init: Y_i ← 𝒴[1]; L_i ← L_1^1

Task T3:
(01) while true do
(02)     Broadcast INQUIRY();
(03)     wait until ( (∃j ∈ Y_i: a corresponding RESPONSE(id_j) is received from p_j)
(04)                  ∨ φ-QUERY(Y_i) ); %  Y_i can dynamically change %
(05)     let rec_from_i = {id_j received previously at line 03 };
(06)     if (rec_from_i ≠ ∅) ∧ (rec_from_i ∩ L_i = ∅) then
(07)                  R_Broadcast L_MOVE(L_i, Y_i) end_if
(08) end_do

Task T4: when L_MOVE(L_i, Y_i) is R_delivered: (L_i, Y_i) ← Next(L_i, Y_i)

Task T5: when INQUIRY() is received from p_j: send RESPONSE(repr_i) to p_j

Task T6: when trusted_i is read by the upper layer:
(09) case   φ-QUERY(Y_i)   then return(min {j : j ∉ Y_i ∧ ¬φ-QUERY(Y_i ∪ {j})})
(10)      ¬φ-QUERY(Y_i)   then return(L_i)
(11) end_case
```

Figure 6: From $\diamond\phi^y + \diamond\mathcal{S}_x$ to Ω^z: upper wheel component (code for p_i)

- Case A. The first case is when all the processes of Y_i have crashed (ϕ-QUERY(Y_i) then eventually returns *true*, lines 04 and 09). It follows that the task $T3$ stops broadcasting *R_Broadcast* L_MOVE(L_i, Y_i) messages. In that case, the value returned for $trusted_i$ (line 09) is the smallest identity among the non-crashed processes.

- Case B. The second case is when p_i receives a response message from a process in Y_i. Then, the set rec_from_i is not empty and contains the identities of the representative $repr_j$ of each process p_j that has answered. We consider two subcases.

  - Case B.1. None of these identities belongs to L_i. p_i then suspects that all the processes of L_i have crashed. It consequently broadcasts the message *R_Broadcast* L_MOVE(L_i, Y_i) to entail the progress of all the processes to the next L_i set.
  - Case B.2. One of these identities belongs to L_i. In that case p_i considers that its current set L_i contains one correct process (the one with that identity). It then continues sending INQUIRY() messages until either none of the identities it receives belongs to L_i (and then we are in case B.1), or it receives a *R_Broadcast* L_MOVE(L_i, Y_i) message which entails its progress to the next L_i.

Let us notice that, due to the property eventually ensured on the $repr_j$ local variables by the lower wheel component, there is a time after which all the RESPONSE(id) messages carry identities of correct processes. It follows that if the set L_i currently investigated by the processes does not change, that set includes at least one correct process and we have obtained the property required for $trusted_i$.

To capture the intuition that underlies the fact that the two wheels synchronize and the processes stabilize on the same set L, let us first recall that, due to the properties of the lower wheel component, there is a time after which there is a set X of x processes such that (1) either all its processes have crashed, or (2) each non-crashed process p_j of X is such that $repr_j = \ell x$ (the identity of a correct process of X). In both cases, a process p_i that does not belong to X is then such that $repr_i = i$.

Let us examine the configuration described in Figure 7. We show that in this configuration a process p_i cannot entail the progress from (L_i, Y_i) to $\mathsf{Next}(L_i, Y_i)$. As this is true for any process p_i, it follows that the processes converge to the same final leader set L_i. The configuration occurs when Y_i contains at least one non-crashed process, $X \subseteq Y_i$, $Y_i \cap X = \{\ell x\}$, and ℓx is the identity of the common representative of the non-crashed processes of X (or the identity of any of them if they all have crashed). In that configuration, any RESPONSE(id) message sent by any process $p_j \in Y_i$ carries an identity that belongs to L_i. It follows that $rec\_from_i \cap L_i \neq \emptyset$ (line 06), and so p_i does not issue *R_Broadcast* L_MOVE(L_i, Y_i) messages.

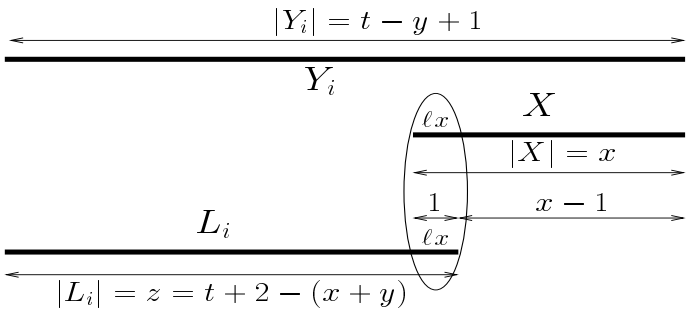


Figure 7: When the upper wheel stops looking for a new L_i set

4.2.2 Proof of the upper wheel component

The proof is very similar to the proof of the lower wheel algorithm. Its structure is the same, and some of its parts are also the same. This is a direct consequence of the fact that both components are based on the same "wheel" principle. The proof considers an arbitrary run of the algorithm. As before, C denotes the set of processes that are correct in that run, and var_i^τ denotes the value of the local variable var_i of at time τ.

Lemma 7 *$\forall i \in C$, there are a pair (Λ_i, Υ_i) and a time τ_i such that $\forall \tau \geq \tau_i : (L_i^\tau, Y_i^\tau) = (\Lambda_i, \Upsilon_i)$.*

Proof We claim (*Claim C2*) that there is a pair (L, Y) such that the number of L_MOVE(L, Y) messages that are sent is finite. Let p_i be a correct process and let us assume (by way of contradiction) that there is no pair (Λ_i, Υ_i) such that after some time $(L_i, Y_i) = (\Lambda_i, \Upsilon_i)$ remains true forever. As the pairs (L, Y) are arranged in a logical ring (as in Figure 4, where $\mathcal{X}[\beta]$ is replaced by $\mathcal{Y}[\beta'], 1 \leq \beta' \leq nb\_y$ and ℓ_α^β is replaced by $L_{\alpha'}^{\beta'}, 1 \leq \alpha' \leq nb\_L$), it follows from the way p_i updates its local pair (L_i, Y_i) that the sequence of the successive values of the local variables (L_i, Y_i) is $(L_1^1, \mathcal{Y}[1]), (L_1^2, \mathcal{Y}[1]), \ldots, (L_{nb\_L}^{nb\_y}, \mathcal{Y}[nb\_y]), (L_1^1, \mathcal{Y}[1])$, etc. Consequently, (L_i, Y_i) takes each values $(L_\alpha^\beta, \mathcal{Y}[\beta]), 1 \leq \alpha \leq nb\_L, 1 \leq \beta \leq nb\_y$ infinitely often. In particular, p_i executes $(L_i, Y_i) \leftarrow \mathsf{Next}(L, Y)$ infinitely often. Since this occurs when p_i delivers a L_MOVE(L, Y) message, this contradicts the Claim *C2* that states that there is a finite number of such messages that are sent. It follows that there are a a pair (Λ_i, Υ_i) and a time τ_i such that $\forall \tau \geq \tau_i : (L_i^\tau, Y_i^\tau) = (\Lambda_i, \Upsilon_i)$.

Claim C2: There is a pair (L, Y) such that the number of L_MOVE(L, Y) messages that are sent is finite.
Proof of Claim C2. We consider two cases according to the number f of actual process crashes.

- Case 1: $f \geq t - y + 1$. Let Y be a set of $t - y + 1$ processes that are faulty and L be an arbitrary subset of Y of size z. Due to the liveness property of the class $\diamond\phi^y$, there is a time τ such that any ϕ-QUERY(Y) issued after time τ by any process always returns *true*. It follows then from lines 06-07 that no message L_MOVE(L, Y) can be broadcast after time τ, which implies that the number of these messages is finite.

- Case: $f < t - y + 1$. Let us consider the time τ at which the lower wheel stops turning. More precisely, it exist a time τ, a set $X \subseteq \Pi, |X| = x$ and a process identity $\lambda \in X$ (Theorem 5) such that:

  1. $\forall i \in \Pi - X, \forall \tau' \geq \tau : repr_i^{\tau'} = i$ and,
  2. (a) $X \cap C \neq \emptyset$: $\exists \lambda \in C \cap X$ such that $\forall i \in X, \forall \tau' \geq \tau : repr_i^{\tau'} = \lambda$ or,
     (b) $X \cap C = \emptyset$: all processes that belong to X have crashed by time τ.

  Let Y be a set of $t - y + 1$ processes identities and L a subset of Y of size z defined as follows (see also Figure 7): (1) $X \subseteq Y$, (2) $|X \cap L| = 1$ and (3) if $X \cap C \neq \emptyset$ then, $X \cap L = \{\lambda\}$. Let $i \in Y$ and let us assume that p_i sends a RESPONSE$(repr_i)$ after time τ. We show that $repr_i \in L$. As $L \cup X \subseteq Y$, $|L| + |X| - 1 = |Y|$ and $|L \cap X| = 1$, either $i \in L$ or $i \in X$. If $i \in X$, it follows from the choice of L that $repr_i = \lambda \in L$. If $i \in L - X$ then, $repr_i = i \in L$. Consequently, it exists a time $\tau' \geq \tau$ after which any message RESPONSE$(repr_i)$ received by any process p_j from a process $p_i, i \in Y$ is such that $repr_i \in L$.

Moreover, as $f < |Y|$, it follows from the eventual safety property of the class $\diamond\phi^y$ that there is a time τ'' from which any ϕ-QUERY(Y) returns *false*.

Hence, after time $\tau''' = \max(\tau', \tau'')$, a process that is waiting for a RESPONSE message from some process p_i that belongs to Y always receives such a message and this message carries a process id $repr_i \in L$. Consequently, it follows from lines 06-07 that after some time τ''', no process can broadcast a L_MOVE(L, Y) message.

End of the Proof of Claim C2. $\square_{Lemma\ 7}$

Corollary 2 *Eventually all processes stop sending* L_MOVE *messages.*

Proof Let us assume (by contradiction) that it exists a correct process p_i that never stop sending L_MOVE messages. Due to Lemma 7, there is a time τ_i after which (L_i, Y_i) remains permanently equal to the constant pair (Λ_i, Υ_i). Consequently, after time τ_i, p_i keeps on broadcasting L_MOVE(Λ_i, Υ_i). It follows then from the validity and termination properties of the reliable broadcast primitive that there is a time $\tau' > \tau_i$ at which p_i executes $(L_i, Y_i) \leftarrow \mathsf{Next}(\Lambda_i, \Upsilon_i)$, contradicting Lemma 7. $\square_{Corollary\ 2}$

Remark. The fact that there is a time after which no L_MOVE(L, Y) messages are exchanged, does not imply that the algorithm is quiescent. This is because the correct processes keep on sending forever INQUIRY() messages, and answering them by sending back RESPONSE() messages. Differently, the lower wheel component uses only X_MOVE() messages.

Lemma 8 $\forall i, j \in C : (\Lambda_i, \Upsilon_i) = (\Lambda_j, \Upsilon_j)$. *(In the following,* (Λ, Υ) *denotes that pair.)*

Proof Due to the properties of the reliable broadcast primitive, p_i and p_j deliver the same multiset of L_MOVE(L, Y) messages. Moreover, it follows from Corollary 2 that this multiset is finite. Due to the fact that p_i and p_j consume the messages according to the same ring order, and the fact that the common multiset of delivered messages is finite, it follows that $(\Lambda_i, \Upsilon_i) = (\Lambda_j, \Upsilon_j)$. $\square_{Lemma\ 8}$

Theorem 6 *The sets* $trusted_i$ *implemented by the algorithm described in Figure 6 satisfy the property defining the class* Ω^z.

Proof Due to Lemma 8, there is a time after which all processes have permanently the same pair (Λ, Υ). We consider two cases:

- $\Upsilon \cap C = \emptyset$. In that case, due to the livened property of the class $\diamond\phi^y$, there is a time after which any ϕ-QUERY(Υ) returns *true*. It follows then from line 09 that all the set $trusted_i$ are eventually equal and contain only the identity of a correct process (namely, the correct process with the lowest identity that does not belong to Υ).

- $\Upsilon \cap C \neq \emptyset$. In that case, any ϕ-QUERY(Υ) eventually always returns *false* (eventual safety property of the class $\diamond\phi^y$). It follows then from line 10 that all the sets $trusted_i$ are eventually permanently equal to Λ. As $|\Lambda| = z$, it remains to show that $\Lambda \cap C \neq \emptyset$.

  Let us assume for contradiction that $\Lambda \cap C = \emptyset$. Let p_i be a correct process. Due to properties ensured by the lower wheel (Theorem 5), there is a time after which any message RESPONSE($repr$) contains the identity of a correct process. From the assumption that Λ contains only faulty processes, it follows that there is a time τ_1 after which p_i cannot receive a RESPONSE message that carries the identity of a process belonging to Λ. Moreover, since set Υ contains at least one correct process, it follows from line 03-04 and the eventual safety property of the class $\diamond\phi^y$ that it exist a time τ_2 after which p_i always gets a RESPONSE($repr_j$) message from some process $p_j, j \in \Upsilon$ while waiting at lines 03-04. Finally, there is a time τ_i after which the predicate $(L_i, Y_i) = (\Lambda, \Upsilon)$ is permanently true (Lemma 7). Consequently, there is a time $\tau \geq \max(\tau_1, \tau_2, \tau_i)$ at which the predicate in the **if** statement of line 06 is not satisfied (i.e., at time τ, we have $rec\_from_i \neq \emptyset \wedge rec\_from_i \cap \Lambda = \emptyset$). It follows then that p_i broadcasts a L_MOVE(Λ, Υ) message. When p_i delivers such a message, it executes $(L_i, Y_i) \leftarrow \mathsf{Next}(\Lambda, \Upsilon)$. The fact that this occurs after the time τ_i contradicts Lemma 7.

$\square_{Theorem\ 6}$

4.3 A Particular Case

Let us examine the case $y = 0$. As already noticed, a failure detector of the class $\diamond\phi^0$ provides no information on failures. In that case, the upper wheel algorithm (Figure 6) can be simplified. As the size of a set Y_i is now $t+1$, ϕ-QUERY(Y_i) returns always *false*. Consequently, the task $T6$ and the line 04 of the task $T1$ can be suppressed, and the value of the set $trusted_i$ provided to the upper layer is the current value of the set L_i.

5 Lower Bounds and (Ir)Reducibility Results

This section states first a lower bound related to the addition of failure detector classes (Figure 2). It then proves the (ir)reducibility results stated in the grid depicted in Figure 1.

5.1 A Lower bound when Adding $\diamond\mathcal{S}_x$ and $\diamond\phi^y$

This section shows that $(x + y + z > t + 1)$ is a lower bound when one wants to add failure detectors of the class $\diamond\mathcal{S}_x$ and $\diamond\phi^y$ to build a failure detector of the class Ω^z.

Theorem 7 *Let us consider any system* $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_x, \diamond\phi^y]$. $(\diamond\mathcal{S}_x + \diamond\phi^y \rightsquigarrow \Omega^z) \Leftrightarrow (x + y + z > t + 1)$.

Proof [$\Leftarrow$ part] This part follows directly from the two wheels algorithm previously described in Sections 4.1.1 and 4.2.1, and proved in sections 4.1.2 and 4.2.2.

[$\Rightarrow$ part] The proof of this part is by contradiction and considers the stronger system $\mathcal{AS}_{n,t}[\mathcal{S}_x, \phi^y]$. As $\mathcal{S}_x \subseteq \diamond\mathcal{S}_x$ and $\phi^y \subseteq \diamond\phi^y$, any impossibility result established in $\mathcal{AS}_{n,t}[\mathcal{S}_x, \phi^y]$ holds in $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_x, \diamond\phi^y]$.

Let us assume that there is an algorithm $\mathcal{T}$ that builds a failure detector of the class Ω^z in $\mathcal{AS}_{n,t}[\mathcal{S}_x, \phi^y]$ with $x + y + z \leq t + 1$. The contradiction is based on the following observations:

- Observation *O1*: Let f be the number of actual failures. When $f \leq t - y$, the only information that a failure detector of the class ϕ^y (or Φ^y) can provide is the fact that the number of failures is $\leq t - y$.
  Proof of O1. Consider a run where $f \leq t - y$. Let $E \subseteq \Pi$. Due to the triviality property of ϕ^y (Φ^y) any ϕ-QUERY(E) returns *true* (resp., *false*) when $|E| \leq t - y$ (resp., $|E| > t$). As $f \leq t - y$ there is always a correct process in any set E such that $t - y < |E| \leq t$. It follows that, due the safety property of ϕ^y (Φ^y), any ϕ-QUERY(E) returns *false* when $t - y < |E| \leq t$. Consequently the boolean value returned by any ϕ-QUERY(E) depends on the size of X, and does not depend on which processes define E. *End of the Proof of O1.*

- Observation *O2*: There is no algorithm that solves the k-set agreement problem in $\mathcal{AS}_{n,t}[\mathcal{S}_x]$ when $t \geq k + x - 1$.
  Proof of O2. This is a lower bound for solving the k-set agreement problem in $\mathcal{AS}_{n,t}[\mathcal{S}_x]$ established in [11]. *End of the Proof of O2.*

Let us now consider the transformation $\mathcal{T}$. In any run where $f \leq t - y$, it follows from *O1* that $\mathcal{T}$ can rely on ϕ^y only to know that the number of failures is $\leq t - y$. This implies that $\mathcal{T}$ can be used to build a failure detector of the class Ω^z in $\mathcal{AS}_{n,t-y}[\mathcal{S}_x]$. Moreover, it exists and algorithm $\mathcal{A}$ that solves the z-set agreement problem in $\mathcal{AS}_{n,t-y}[\Omega^z]$ (such an algorithm is described in Section 3). Consequently, by combining transformation $\mathcal{T}$ and algorithm $\mathcal{A}$, one can solve the z-set agreement problem in $\mathcal{AS}_{n,t-y}[\mathcal{S}_x]$. Hence, it follows from *O2* that the constraint $t - y < z + x - 1$ has to be satisfied, from which we obtain $x + y + z > t + 1$: a contradiction. $\square_{Theorem\ 7}$

The following corollary is an immediate consequence of the proof of Theorem 7.

Corollary 3 *Let us consider any system* $\mathcal{AS}_{n,t}[\mathcal{S}_x, \phi^y]$, $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_x, \phi^y]$ *or* $\mathcal{AS}_{n,t}[\mathcal{S}_x, \diamond\phi^y]$. *In any of these systems, it exists an algorithm that builds a failure detector of the class* Ω^z *if and only if* $(x + y + z) > t + 1$.

The following corollary follows directly from the proof of the necessity ($\Rightarrow$) part of Theorem 7.

Corollary 4 *Let us consider any system* $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_x, \Phi^y]$. $(\diamond\mathcal{S}_x + \Phi^y \rightsquigarrow \Omega^z) \Rightarrow (x + y + z > t + 1)$.

The following corollary is another consequence of Theorem 7.

Corollary 5 *The two wheels algorithm described in Figures 5 and 6 is optimal with respect to the possible values of* x, y *and* z.

As $\diamond\mathcal{S}_1$ (case $x = 1$) provides no information on failures, we directly obtain the following corollary from the two wheel algorithm and Theorem 7.

Corollary 6 *It is possible to build a failure detector of the class* Ω^z *in* $\mathcal{AS}_{n,t}[\phi^y]$ *or* $\mathcal{AS}_{n,t}[\diamond\phi^y]$ *if and only if* $y + z > t$.

Similarly, as $\diamond\phi^0$ (case $y = 0$) provides no information on failures, we also have:

Corollary 7 *It is possible to build a failure detector of the class* Ω^z *in* $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_x]$ *if and only if* $x+z > t+1$.

5.2 Relations between $\mathcal{S}_x/\diamond\mathcal{S}_x$ and $\phi^y/\diamond\phi^y$

Theorem 8 *Let* $1 \leq x \leq t+1$ *and* $1 \leq y \leq t$. *It is not possible to build a failure of the classes* ϕ^y ,$\diamond\phi^y$ *(or* Φ^y*) neither in* $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_x]$ *nor in* $\mathcal{AS}_{n,t}[\mathcal{S}_x]$.

Proof The proof considers the "stronger" system $\mathcal{AS}_{n,t}[\mathcal{S}_x]$. As $\mathcal{S}_x \subseteq \diamond\mathcal{S}_x$, the proof remains valid for a system $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_x]$. Similarly, as $\Phi^y \subset \phi^y \subseteq \diamond\phi^y$ the proof considers only the "weaker" class $\diamond\phi^y$ in the following. The proof is by contradiction. Let us assume that there is a failure detector $\mathcal{F}$ of the class $\mathcal{S}_x$ and an algorithm $\mathcal{A}$ that transforms $\mathcal{F}$ into a failure detector of the class $\diamond\phi^y$. We exhibit a run R in which the eventual safety property of the class $\diamond\phi^y$ is not satisfied.

Let $E \subseteq \Pi$, $|E| = t - y + 1$ and $E \cap C \neq \emptyset$. Let p_c be a correct process that does not belong to set E. Moreover, p_c is never suspected by $\mathcal{F}$ in run R. Let τ_0 be the time at which any ϕ-QUERY(E) invoked after time τ_0 returns the value *false*. Such a time exists due to the correctness of algorithm $\mathcal{A}$ and the eventual safety property of the class $\diamond\phi^y$. We consider the following runs $R1$ and $R1'$ which are possible suffix of run R after time τ_0:

- Runs $R1$ and R are indistinguishable by all processes until time τ_0. A time $\tau_0 + 1$, all processes that belong to E crash. Let $\tau_1 > \tau_0$ be a time at which a process $p_i \in \Pi - E$ invokes ϕ-QUERY(E) and obtains the value *true*. Such a time must exist due to liveness property of the class $\diamond\phi^y$.
- Runs $R1'$ and R are indistinguishable by all processes until time τ_0. In the run $R1'$, all the processes in E are correct, but all the messages they send between times $\tau_0 + 1$ and τ_1 are delayed until time $\tau_1 + 1$.

Moreover, both runs are such that the outputs of the failure detector $\mathcal{F}$, at each process, are exactly the same between the times 0 and τ_1. (Let us notice that whatever be the output of $\mathcal{F}$ in $R1$, the output of $\mathcal{F}$ can be exactly the same in $R1'$ without violating the properties of the class $\mathcal{S}_x$. As p_c is correct in $R1$ and $R1'$ and never suspected in $R1$ and $R1'$, limited scope perpetual accuracy is insured. Since strong completeness is an eventual property, it is always satisfied in any finite prefix of any execution.) Clearly, up to time τ_1, the processes that belong to $\Pi - E$ cannot distinguish the run $R1$ from the run $R1'$. It follows that, in the run $R1'$, an invocation of ϕ-QUERY(E) by p_i at time $\tau_1 > \tau_0$ returns the value *true*. But in run $R1'$, a ϕ-QUERY(E) issued after time τ_0 must return the value *false*: a contradiction. $\square_{Theorem\ 8}$

Theorem 9 *Let* $0 \leq y < t$ *and* $1 < x \leq t + 1$. *It is not possible to build a failure of the class* $\mathcal{S}_x$ *or* $\diamond\mathcal{S}_x$ *neither in* $\mathcal{AS}_{n,t}[\diamond\phi^y]$ *nor in* $\mathcal{AS}_{n,t}[\phi^y]$ *(nor in* $\mathcal{AS}_{n,t}[\Phi^y]$*).*

Proof Let us first notice that we need to prove only the impossibility to build a failure detector of the class $\diamond\mathcal{S}_x$ in $\mathcal{AS}_{n,t}[\phi^y]$. The proof is by contradiction and uses the following observations.

- Observation *O1*: Let f be the number of actual failures. When $f \leq t-y$, the only information that a failure detector of the class ϕ^y (or Φ^y) can provide is the fact that the number of failures is $\leq t-y$. (This observation has already been stated and proved in Theorem 7.)

- Observation *O2*: There are algorithms that solve the k-set agreement problem in $\mathcal{AS}_{n,t}[\mathcal{S}_x]$. All these algorithms require $t \leq k+x-2$. (Examples of such algorithms can be found in [11, 19]. The lower bound on t is established in [11].)

- Observation *O3*: The k-set agreement problem can be solved in $\mathcal{AS}_{n,t-y}[\emptyset]$ if and only if $k > t$. (The proof of this observation constitutes an important result of fault-tolerant distributed computing. It can be found in [1, 12, 23].)

Let us suppose that there is an algorithm $\mathcal{A}$ that builds a failure detector of the class $\diamond\mathcal{S}_x$ from a failure detector of the class ϕ^y (Φ^y). In any run where $f \leq t-y$, it follows from *O1* that $\mathcal{A}$ can rely on ϕ^y (Φ^y) only to know that the number of failures is $\leq t-y$. Consequently, $\mathcal{A}$ can build a failure detector of the class $\diamond\mathcal{S}_x$ in a system $\mathcal{AS}_{n,t-y}[\emptyset]$. This means that one can use $\mathcal{A}$ to solve the the $(t-y)-x+2$-set agreement problem using any algorithm listed in observation $O2$ in a system $\mathcal{AS}_{n,t-y}[\emptyset]$. We then conclude from $O3$ that $(t-y)-x+2 > t-y$, i.e., $x \leq 1$, a contradiction with the assumption $1 < x \leq n$ [2].

$\square_{Theorem\ 9}$

5.3 From Ω^z to $\phi^y/\diamond\phi^y$ or $\mathcal{S}_x/\diamond\mathcal{S}_x$

It has been shown (Corollaries 7 and 6) that it is possible to build a failure detector of the class Ω^z from any failure detector of the classes $\mathcal{S}_x/\diamond\mathcal{S}_x$ (resp., $\phi^y/\diamond\phi^y$) if and only if $x+z > t+1$ (resp., $y+z > t$). This section shows that it is not possible to build a failure detector of the classes $\mathcal{S}_x/\diamond\mathcal{S}_x$ (resp., $\phi^y/\diamond\phi^y$) from any failure detector of the class Ω^z. The proofs of these impossibilities derived from Theorem 9 and 8.

Theorem 10 *Let $1 \leq y \leq t$ and $1 \leq z \leq t+1$. It is impossible to build a failure detector of a class $\phi^y/\diamond\phi^y$ (or Φ^y) in $\mathcal{AS}_{n,t}[\Omega^z]$.*

Proof The proof is by contradiction. Let us assume that there is an algorithm $\mathcal{A}$ that builds a failure detector of a class $\diamond\phi^y$, $1 \leq y \leq t$, from any failure detector of a class Ω^z, $1 \leq z \leq t+1$. Due to Corollary 7, it is possible to build a failure detector of a class Ω^z in $\mathcal{AS}_{n,t}[\diamond\mathcal{S}_x]$ when $x+z > t+1$, i.e., when $z > t-x+1$. Combining this construction with the algorithm $\mathcal{A}$ we obtain an algorithm $\mathcal{B}$ that builds a failure detector of the class ϕ^y, $1 \leq y \leq t$ from a failure detector of the class $\diamond\mathcal{S}_x$ when $t+1-z < x \leq t+1$ and $1 \leq z \leq t+1$. But such an algorithm $\mathcal{B}$ contradicts Theorem 8 that states that there is no such algorithm when $1 \leq x \leq t+1$ and $1 \leq y \leq t$. $\square_{Theorem\ 10}$

Theorem 11 *Let $1 < x, z \leq t$. It is impossible to build a failure detector of the class $\mathcal{S}_x/\diamond\mathcal{S}_x$ in $\mathcal{AS}_{n,t}[\Omega^z]$.*

Proof The proof is similar to the proof of Theorem 10. It is left to the reader. $\square_{Theorem\ 11}$

5.4 Optimality in the Grid

It follows from all the previous theorems and lemmas that, when we consider all the failure detector classes depicted in Figure 1, Ω^k is the weakest class that allows solving the k-set agreement problem. This constitutes a first step towards the characterization of the weakest failure detector class for solving that problem. A corresponding Ω^k-based k-set agreement protocol has been described in Section 3.

[2] Let us remind that the failure detectors of the classes $\mathcal{S}_1$ and $\diamond\mathcal{S}_1$ provide no information on failures.

References

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

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

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

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

[5] Chu F., Reducing Ω to $\diamond\mathcal{W}$. *Information Processing Letters*, 76(6):293-298, 1998.

[6] Dutta P. and Guerraoui R., Fast Indulgent Consensus with Zero Degradation. *Proc. 4th European Dependable Computing Conference (EDCC'02)*, Springer-Verlag LNCS #2485, pp. 191-208, 2002.

[7] Delporte-Gallet C., Fauconnier H. and Guerraoui R., (Almost) All Objects are Universal in Message Passing Systems. *Proc. 19th Symposium on Distributed Computing (DISC'05)*, Springer Verlag LNCS #3724, pp. 184-198, 2005.

[8] Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.

[9] Guerraoui R. and Raynal M., The Information Structure of Indulgent Consensus. *IEEE Transactions on Computers.* 53(4), 53(4):453-466, 2004.

[10] Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems*, ACM Press, New-York, pp. 97-145, 1993.

[11] Herlihy M.P. and Penso L. D., Tight Bounds for k-Set Agreement with Limited Scope Accuracy Failure Detectors. *Proc. 17th Symposium on Distributed Computing (DISC'03)*, Springer Verlag LNCS #2848, pp. 279-291, 2003.

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

[13] Guerraoui R., Non-Blocking Atomic Commit in Asynchronous Distributed Systems with Failure Detectors. *Distributeed Computing*, 15:17-25,2002.

[14] Lamport L., The Part-Time Parliament. *ACM Transactions On Computer Systems*, 16(2):133-169, 1998.

[15] Mostefaoui A., Rajsbaum S. and Raynal M., Conditions on Input Vectors for Consensus Solvability in Asynchronous Distributed Systems. *Journal of the ACM,* 50(6):922-954, 2003.

[16] Mostefaoui A., Rajsbaum S. and Raynal M., The Combined Power of Conditions and Failure Detectors to Solve Asynchronous Set Agreement. *Proc. 24th ACM Symposium on Principles of Distributed Computing (PODC'05)*, ACM Press, pp. 179-188, 2005.

[17] Mostefaoui A., Rajsbaum S., Raynal M. and Travers C., From $\diamond\mathcal{W}$ to Ω: a Simple Bounded Quiescent Reliable broadcast-based Transformation. *Tech Report #xyz*, IRISA, University of Rennes 1 (France), 2005.

[18] Mostéfaoui A. and Raynal M., Solving Consensus Using Chandra-Toueg's Unreliable Failure Detectors: a General Quorum-Based Approach. *Proc. 13th Symposium on Distributed Computing (DISC'99)*, Springer Verlag LNCS #1693, pp. 49-63, 1999.

[19] Mostefaoui 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.

[20] Mostéfaoui A. and Raynal M., Leader-Based Consensus. *Parallel Processing Letters*, 11(1):95-107, 2001.

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

[22] Raynal M., A Short Introduction to Failure Detectors for Asynchronous Distributed Systems. *ACM SIGACT News, Distributed Computing Column*, 36(1):53-70, 2005.

[23] 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.

[24] Schiper A., Early Consensus in an Asynchronous System with a Weak Failure Detector. *Distributed Computing*, 10:149-157, 1997.

[25] Yang J., Neiger G. and Gafni E., Structured Derivations of Consensus Algorithms for Failure Detectors. *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC'98)*, pp.297-308, 1998.

A A simple construction from Φ^y to Ω^z $(y + z > t)$

Assuming $y + z > t$, the algorithm described in Figure 8 builds a failure detector of the class Ω^z from any failure detector of the class Φ^y. Let us remind that the class Φ^y is ϕ^y with the additional requirement that any two sets $X1$ and $X2$ used as query parameter must be such that $X1 \subseteq X2$ or $X2 \subseteq X1$.

The transformation is extremely simple. There is a sequence of $n - z + 2$ sets (initially known by all the processes), defined as follows. Each set contains process identities, and we have: $Y[0] = \emptyset$; $Y[1]$ is an arbitrary set of z distinct process identities; and for any $1 < i \leq n - z + 1$, $Y[i-1] \subset Y[i]$ and $|Y[i-1]| + 1 = |Y[i]|$. Let us observe that any two sets satisfy the containement property, and $Y[n - z + 1] = \Pi$.

The value of the set $trusted_i$ provide to the upper layer is computed as follows. Using the Φ-QUERY() primitive p_i determines the first set $Y[k]$ in the previous sequence that contains a process that has not crashed. As $z > t - y$, we have $|Y[m]| = z > t - y$ for $m > 0$. This means all the processes included in the previous sets have crashed. The processes p_i returns then the set $Y[k] \setminus Y[k+1]$.

```
when trusted is read by the upper layer:
    let k = min{j : ¬Φ-QUERY(Y[j])};
    return(Y[k] \ Y[k − 1])
```

Figure 8: From Φ^y to Ω^z $(y + z > t)$

Theorem 12 *There a finite time after which the sets $trusted_i$ of all the correct processes are equal, of size at most z, and contain one correct process.*

Proof Let C the set of correct processes, $|C| \geq 1$. Let $m = \min\{j : Y[j] \cap C \neq \emptyset\}$. Let us observe that m exists. This is because $Y[n - z + 1] = \Pi$ and $|C| \geq 1$. If follows that m is such that $Y[m]$ is the first set of the sequence $Y[0], \ldots, Y[n - z + 1]$ that contains the identity of a correct process. Let us observe that $m > 0$ since $Y[0] = \emptyset$. From the properties defining the class Φ we have:
- As $Y[0] = \emptyset$, Φ-QUERY$(Y[0])$ returns always *true* (triviality property of Φ).
- There is a time after which all the invocations of Φ-QUERY(A) where A is $Y[a]$ with any $a < m$ returns *true* (liveness property of Φ).
- All the invocations Φ-QUERY$(Y[m])$ returns *false* (safety property of Φ).
It follows that there is a time after which all the correct processes output $Y[m] \setminus Y[m-1]$, which proves the theorem. (If one process of $Y[1]$ is correct, they output $Y[1]$ of size z; otherwise they output a single process identity.) $\square_{Theorem\ 12}$

B A simple addition $\diamond\mathcal{S}_x + \diamond\phi^y \rightsquigarrow \diamond\mathcal{S}_n$ $(x + y > t)$

This section presents a simple algorithm that adds the power of ϕ^y and the power of $\mathcal{S}_x$ (resp., $\diamond\phi^y$ and $\diamond\mathcal{S}_x$) to provide a failure detector of the class $\mathcal{S}_n$ (resp., $\diamond\mathcal{S}_n$). (Let us remind that $\mathcal{S}_n = \mathcal{S}$ and $\diamond\mathcal{S}_n = \diamond\mathcal{S}$.) The algorithm is described in Figure 9. As the failure detector classes $\Omega^1 = \Omega$ and $\diamond\mathcal{S}_n = \diamond\mathcal{S}$ are equivalent

(they have the same computational power as far as failures are concerned) [2, 5, 17], it follows from Theorem 7 that the algorithm requires $x + y > t$ (which becomes a necessary and sufficient requirement for such a transformation).

To show the versatility of the approach, the algorithm is expressed in the shared memory model. It can be easily translated in the message-passing model without adding any requirement on t. Each process p_i has the following local variables:

- $suspected_i$ is a local variable that p_i can only read. It contains the set of processes provided to p_i by its underlying failure detector module of the class $\mathcal{S}_x$ (resp., $\diamond\mathcal{S}_x$). These sets satisfy the properties defining the class $\mathcal{S}_x$ (resp., $\diamond\mathcal{S}_x$): they eventually includes all crashed processes and x of these sets do not include the same correct process from the very beginning in the case of $\mathcal{S}_x$ (or after some unknown but finite time in the case of $\diamond\mathcal{S}_x$).

- $SUSPECTED_i$ is the local set of processes built by the algorithm. The sets $SUSPECTED_i$ of all the processes have to satisfy the properties defining $\mathcal{S}$ (resp., $\diamond\mathcal{S}$). Initially, $SUSPECTED_i = \emptyset$.

- new_i and $prev_i$ are two auxiliary variables. Each is an array of size n initialized to the zero vector.

The shared memory is made up of two arrays denoted $alive[1:n]$ and $suspect[1:n]$. Each of their entries is a single writer/multi reader atomic variable. The $alive[i]$ and $suspect[i]$ variables are repeatedly updated by p_i until it (possibly) crashes (see task $T1$ in Figure 9). Their meaning is the following:

- $alive[i]$ is only increased by p_i to indicate it has not crashed. This means that, after a process p_i crashes, the value of $alive[i]$ does not change[3].

- $suspect[i]$ is used by p_i to inform the other processes about the value of its local $suspected_i$ set.

The task $T2$ of a process p_i repeats forever a set of statements whose aim is to compute the current value of the local set $SUSPECTED_i$ (line 07) whose value is used by the upper layer protocol. To carry out this computation, p_i first reads the shared array $alive[1:n]$ to know which processes have progressed (the reading of the whole array is not atomic). It reads this array until it knows that all the processes that have not progressed have crashed (lines 02-05). Then, trusting the processes it considers as not crashed (the set $live$), it updates its local set $SUSPECTED_i$ according to the current suspicions made public by these processes.

```
task T1: repeat forever
(01)            alive[i] ← alive[i] + 1; suspect[i] ← suspected_i
         end repeat

task T2: repeat forever
(02)            repeat new_i ← [read(alive[1]),···,read(alive[n])];
(03)                   let live = {j | new_i[j] > prev_i[j]};
(04)                   let X = {1, ..., n} \ live;
(05)            until φ-QUERY (X);
(06)            prev_i ← new_i;
(07)            SUSPECTED_i ← (⋂_{j∈live} suspect[j]) \ live
         end repeat
```

Figure 9: From $\phi^y + \mathcal{S}_x$ to $\mathcal{S}$ (resp., $\diamond\phi^y + \diamond\mathcal{S}_x$ to $\diamond\mathcal{S}$), (algorithm for p_i)

Theorem 13 *Let $x + y > t$. If the underlying failure detector belongs to the class $\mathcal{S}_x$ (resp., $\diamond\mathcal{S}_x$), the sets $SUSPECTED_i$ built by the ϕ^y-based (resp., $\diamond\phi^y$-based) algorithm described in Figure 9 define a failure detector of the class $\mathcal{S}$ (resp., $\diamond\mathcal{S}$).*

[3] It is possible to have a bounded implementation for each shared variable $alive[i]$. We do not elaborate on this for two reasons: on one side it would make the protocol much more involved, on another side this is not necessary to prove our goal.

Proof Let us first show that the inner loop always terminates. Proving this termination is required to claim that the variable $SUSPECTED_i$ is updated at line 07. We consider three cases according to the size of the set parameter X when p_i invokes QUERY (X) at line 05.

- $|X| > t$. In that case, due to the triviality property, the query returns *false*, and p_i enters again the loop. But, as there are at most t faulty processes, each correct process p_j infinitely often increases $alive[j]$ (task $T1$), and $prev_i[j]$ remains constant within the inner loop, there is a time after which every query issued by p_i is such that $|X| \leq t$. We are then in one of the cases that follow.

- $|X| \leq t - y$. In that case, due to the triviality property, the query returns *true* and p_i exits the inner loop.

- $t - y < |X| \leq t$. If the query returns *false*, p_i enters again the loop. We show that the query eventually returns *true*. Let us consider a process p_j that belongs to S (this means that $alive[j] = prev_i[j]$ at line 03 of the task $T2$ executed by p_i). If p_j is correct, there is eventually an inner loop such that $alive[j] > prev_i[j]$ because p_j increases forever $alive[j]$ and $prev_i[j]$ remains constant within the inner loop. This means that eventually such a p_j will disappear from the set X defining the query parameter. It follows that, eventually, the set X used as a query parameter (1) contains only faulty processes or (2) has a size smaller than or equal to $t - y$. Due to the liveness (case 1) or triviality (case 2) property, there is then a query that eventually returns *true*.

Let us now show that, if the sets $suspected_i$ satisfy the strong completeness property, this property is also satisfied by the sets $SUSPECTED_i$. If a process p_k crashes, due to the strong completeness of the sets $suspected_i$, it eventually belongs to the set $suspected_j$ of each non-crashed process p_j. Due to line 01, after some finite time, p_k is always in $suspect_j$ (until p_j possibly crashes). Moreover, as after some time p_k no longer increases $alive[k]$, there is a finite time after which it never belongs to the *live* set computed by any process. Due to line 07, it eventually belongs to (and remains permanently in) the set $SUSPECTED_i$ of any non-crashed process p_i.

The last part of the proof concerns the weak accuracy property. We formulate the proof for going from the class $\mathcal{S}_x$ to the classes $\mathcal{S}$. (The proof for going from the class $\diamond\mathcal{S}_x$ and $\diamond\phi^y$ to the class $\diamond\mathcal{S}$ is similar, and is consequently omitted.) So, we have to show that, if $x + y > t$ and the sets $suspected_i$ satisfy the limited scope perpetual weak accuracy property (namely, there is a correct process, say p_ℓ, that is not suspected by at least x -correct or faulty- processes), then the sets $SUSPECTED_i$ satisfy perpetual weak accuracy (there is a correct process -namely, p_ℓ again in our transformation- that is no suspected by any process). We consider two cases, according to the size of the set X when a process p_i exits the inner loop.

- $|X| \leq t - y$.
  In that case, the exit of the inner loop was due to the triviality property. As $t - y < x$, we have $|X| < x$, which (due the limited scope perpetual weak accuracy) means that at least one process p_k of the set *live* of p_i is such that p_ℓ never belongs to $suspected_k$, and consequently p_ℓ never belongs to $suspect[k]$. It then follows that p_ℓ can never belong to the intersection computed at line 07, which proves the case.

- $t - y < |X| \leq t$.
  In that case, due to the safety property, all the processes in X have crashed. We examine two subcases.
  - $t - y < |X| < x$. The proof of this case ($|X| < x$) is the same as the previous one.
  - $t - y < x \leq |X|$. In that case, it is possible that all the processes that do not suspect p_ℓ have crashed, and all the remaining processes p_j do suspect p_ℓ (i.e., $p_\ell \in suspected_j$). But in that case (noticing that X and *live* define a partition of the whole set of processes), a process that is not in the *live* set of p_i has necessarily crashed (safety and non-triviality properties). So, p_ℓ necessarily belongs to the set *live* of p_i. It follows from line 07 that p_ℓ cannot belong to $SUSPECTED_i$, which proves the case.

□$_{Theorem\ 13}$