Sharing is Harder than Agreeing

Carole Delporte-Gallet
LIAFA Université Paris 7 - Denis Diderot
75205 Paris Cedex 13, France
cd@liafa.jussieu.fr

Hugues Fauconnier
LIAFA Université Paris 7 - Denis Diderot
75205 Paris Cedex 13, France
hf@liafa.jussieu.fr

Rachid Guerraoui
EPFL
CH-1015 Lausanne, Switzerland
rachid.guerraoui@epfl.ch

ABSTRACT

One of the most celebrated results of the theory of distributed computing is the impossibility, in an asynchronous system of n processes that communicate through shared memory registers, to solve the set agreement problem where the processes need to decide on up to $n-1$ among their n initial values. In short, the result indicates that the register abstraction is too weak to implement the set agreement one.

This paper explores the relation between these abstractions in a message passing system where a register is not a given physical device but is rather itself implemented by processes communicating through message passing. We show that, maybe surprisingly, the information about process failures that is necessary and sufficient to implement a register shared by two particular processes is sufficient but not necessary to implement set agreement.

We later generalize this result by considering k-set agreement, where the processes can decide on up to k values, and comparing it with a register shared by any particular subset of $2k$ processes. We prove that, for $1 \leq k \leq n/2$, (a) any failure information that is sufficient to implement a register shared by $2k$ processes is sufficient to implement $(n-k)$-set agreement but (b) a failure information that is sufficient for $(n-k)$-set agreement is not sufficient for a register shared by 2k processes. We also prove that (c) a failure information that is sufficient for a register shared by 2k processes is not sufficient for ((n-k)-1)-set agreement.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Network**]: Distributed Systems—*distributed applications, network operating systems*; D.4.1 [**Operating Systems**]: Process Management—*concurrency, multiprocessing, synchronization*; D.4.5 [**Operating Systems**]: Reliability—*fault-tolerance*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Computability theory*

General Terms

Algorithms, Reliability, Theory

Keywords

Asynchronous system, Message passing, k-Set agreement, Register, Failure detector

1. INTRODUCTION

One of the fundamental results of the theory of distributed computing, proved by three independent groups of researchers [21, 13, 3] and celebrated by the Gödel prize, is the impossibility, in an asynchronous shared memory system, of implementing *set agreement* [6]: a decision task where n processes, starting each with an initial value, need to agree on up to $n-1$ among their n initial values. In short, the result conveys the fact that the basic read/write shared memory abstraction, also called the *register* [15] abstraction, is not powerful enough to help the processes eliminate one of their initial values, if these processes can run at their own speed, including stopping without warning, i.e., failing by crashing.

In many cases however, a register is not available as a physical device accessible to the processes but is rather implemented (we also say *emulated* [1]) in a distributed system where the processes communicate solely by exchanging messages. Such an implementation cannot be achieved however without any information about process failures, e.g., only a minority of processes can crash [1] or all failures are accurately detected [5].

At first glance, one would expect the failure information that is necessary and sufficient to implement a register not to be sufficient to implement set agreement. This paper shows that, may be surprisingly, it is actually sufficient. Moreover, we show that the failure information that is necessary and sufficient to implement set agreement is not sufficient to implement a register. We establish our results precisely by expressing the notion of *failure information* using the *failure detector* formalism of [5].[1] We show that (a.1) any failure detector that implements a register shared by (a specific pair of) two processes also implements set agreement, but (b.1) not vice-versa.

The intuition behind (a.1) lies in the fact that the failure information that is needed to implement a register shared by two processes is sufficient for these processes to eliminate at least one of their initial values (and thus for the entire set of processes to implement set agreement). The intuition

[1]We consider a system of at least three processes. In a system of two processes, the two abstractions are equivalent [9].

2-register	$\rightarrow$	set agreement
	$\not\leftarrow$	set agreement
3-register	$\not\rightarrow$	$(n-2)$-set agreement
$\vdots$		
$2k$-register	$\rightarrow$	$(n-k)$-set agreement
	$\not\leftarrow$	$(n-k)$-set agreement
$2k+1$-register	$\not\rightarrow$	$(n-k-1)$-set agreement

Figure 1: Results

behind (b.1) is that, to eliminate one of the initial values of a set of processes, it is enough for at least one process to learn that some other process is correct, without knowing which one. This is however not enough to implement a register abstraction, even shared only among two processes. We capture these intuitions more specifically in the paper, by introducing a failure detector that might be of independent interest. This failure detector, denoted σ, chooses exactly two processes and distinguish them from the rest. If these two processes are the only correct ones in the system then σ also provides them with information about *correct quorums* of processes and this is used to eventually eliminate at least one of their values. We prove that σ is sufficient to implement set agreement but not a register.

Our result naturally raises the more general question of the relation between implementing a register and implementing k-set agreement, where the goal is for the processes to agree on up to $k \leq (n-1)$ values, i.e., to eliminate $n-k$ values. Interestingly, the failure information that is necessary and sufficient to implement a register shared by two processes is not sufficient to implement $(n-2)$-set agreement. More generally, assuming $1 \leq k \leq n/2$, we prove that (Figure 1) (a.2) any failure detector that implements a register shared by (a specific subset) of $2k$ processes implements $(n-k)$-set agreement; (b.2) a failure detector might be sufficient to implement $(n-k)$-set agreement but not a register shared by $2k$ processes; and finally (c) a failure detector might be sufficient to implement a register shared by $2k$ processes but not to implement $((n-k)-1)$-set agreement.

The rest of the paper is organized as follows. Section 2 defines the basic model of computation and recalls the notions of X-register ([9]), Σ_X ([9]) and k-set agreement ([6]). Section 3 shows that implementing a register shared by two processes is strictly harder than implementing set agreement. Section 4 generalizes this result and shows that implementing a register shared by $2k$ processes is strictly harder than implementing $(n-k)$-set agreement. Section 5 completes the results of Sections 3 and 4 by showing that implementing a register shared by $2k+1$ processes is not harder than implementing $(n-(k+1))$-set agreement. Section 6 discusses related work.

2. MODEL AND DEFINITIONS

2.1 Processes, failures and failure detectors

Our model of computation is the one of [4]. In this section, we simply recall elements of the model that are needed to state and prove our results.

The system we consider is a set Π of n processes that communicate through reliable channels. The processes run asynchronously at their own speed and can fail by crashing. A process that does not fail in a given run (we recall what a run is below) is said to be correct. Each run of the processes is associated with a failure pattern: a function F that associates, to each time $t \in \Phi$ (Φ is a global clock that can not be accessed by the processes), the set of processes that have crashed by time t. We denote by $Correct(F)$ the set of correct processes in F. An environment is a set of failure patterns. We consider in this paper failure patterns where at least one process is correct, and, except when explicitly stated, we focus on the environment, denoted by $\mathcal{E}$, that contains all failure patterns with at least one correct process. A failure detector history H with range R is a function that associates, to each process p and each time t, a value $H(p,t)$ in R. A failure detector $\mathcal{D}$ with range R is a function that associates, to each failure pattern, a set of failure detector histories $\mathcal{D}(F)$ with range R.

A distributed algorithm A using a failure detector $\mathcal{D}$ is a collection of n deterministic automata (one per process in the system). A run (execution) of A occurs in steps: for every time $t \in \Phi$, at most one process takes a step; every correct process takes an infinite number of steps. For any given failure pattern F and any given failure detector history H of $\mathcal{D}(F)$, in each step, a process p atomically performs the following three actions: (1) p receives a message from some process or a null message, (2) p queries and receives a value $H(p,t)$ from its failure detector module, and (3) p changes its state and sends a message (possibly null) to some process.

Every abstraction U (e.g., *register, k-set-agreement*, etc) is associated with exactly one set of runs, i.e., the runs that obey the properties of U. We say that an algorithm A implements U using a failure detector $\mathcal{D}$ if every run of A using $\mathcal{D}$ is in U. We say that $\mathcal{D}$ implements U if there is an algorithm that implements U using $\mathcal{D}$.

A failure detector $\mathcal{D}'$ is said to be stronger than a failure detector $\mathcal{D}$ (we write $\mathcal{D} \preceq \mathcal{D}'$) if there is an algorithm that emulates the output of $\mathcal{D}$ (we also say implements $\mathcal{D}$) using $\mathcal{D}'$. Two failure detectors are equivalent if each is stronger than the other. If $\mathcal{D}'$ is stronger than $\mathcal{D}$ and they are not equivalent then we say that $\mathcal{D}'$ is strictly stronger than $\mathcal{D}$ (we write $\mathcal{D} \prec \mathcal{D}'$). We say that a failure detector $\mathcal{D}$ is the weakest to implement an abstraction U if (a) $\mathcal{D}$ implements U and (b) any failure detector that implements U is stronger than $\mathcal{D}$.

We say that an abstraction U is harder than an abstraction U' if (1) the weakest failure detector to implement U implements U'. We say that an abstraction U is strictly harder than an abstraction U' if (1) U is harder than U' and (2) U' is not harder than U.

2.2 Registers

A register is a shared object accessed through two operations: read and write. The write operation takes as an input parameter a specific value to be stored in the register and returns a simple indication *OK* conveying the fact that the operation has been executed. In the absence of concurrency, the read operation is supposed to return the last value written in the register.

We consider *atomic* [15], also called *linearizable* [14], registers. Roughly speaking, these ensure that, despite concurrent invocations and possible crashes of the processes, every correct process that invokes an operation eventually gets a reply (a value for the read and an OK indication for the write), and every operation appears to be executed instantaneously between its invocation and reply time events [15, 14, 2].

Given a subset S of processes in Π, a *S-register* ([9]) is a register that can be read and written only by processes in S. When S is the overall set Π of processes, such a register (i.e., that can be read and written by any process) is called a (*multi-writer/multi-reader*) register [15]. By language abuse, given an integer $1 \leq k \leq n$, we use the term k-register to denote a X-register for some specific subset X of k processes in Π.

Failure detector Σ_S [9] outputs, at each process of S, and at any time, a list of processes called *trusted* processes, such that the following properties are ensured: (these properties assume that, at any process of S that has crashed, the list that is output is Π)

- *Intersection.* Every two lists of trusted processes intersect:
  - $\forall F \in \mathcal{E}, \forall H \in \Sigma_S(F), \forall p, q \in S, \forall t, t' \in \Phi : H(p,t) \cap H(q,t') \neq \emptyset$
- *Completeness.* Eventually, every list of processes trusted by every correct process contains only correct processes:
  - $\forall F \in \mathcal{E}, \forall H \in \Sigma_S(F), \forall p \in S \cap Correct(F), \exists t \in \Phi, \forall t' > t \in \Phi : H(p,t') \subseteq Correct(F)$

It is easy to see that, for any subset S, failure detector Σ_S can be implemented (with no synchrony assumption) in any environment where a majority of processes is correct. Every process periodically sends a message to all, asking for replies, waits for a majority of these, and outputs the list of processes which indeed replied.

We recall the result of [9]:

PROPOSITION 1. *In any environment, for any subset S of processes, Σ_S is the weakest failure detector to implement a S-register.*

2.3 The k-set agreement problem

Given a positive integer k, solving k-set agreement [6] consists for every process $p \in \Pi$, starting with some initial value v_p, to satisfy the following properties: *1. Agreement:* At most k different values are decided; *2. Termination:* every correct process eventually decides; *3. Validity:* if any process decides a value v, then v is the initial value of some process .

Sometimes, we simply write set agreement for $(n-1)$-set agreement.

3. A (2-)REGISTER IS STRICTLY HARDER THAN SET AGREEMENT

We prove in this section the following:

THEOREM 2. *$\forall n \geq 3, \forall p, q \in \Pi$, a $\{p,q\}$-register is strictly harder than set agreement.*

To prove this theorem we first introduce a new failure detector, σ, and we prove that it implements set agreement (Sec. 3.1). Then we show that $\Sigma_{\{p,q\}}$ is stronger than σ, proving that a $\{p,q\}$-register is harder than set agreement (Sec. 3.2). Finally we prove that σ is not stronger than $\Sigma_{\{p,q\}}$, deducing therefore that set agreement is not harder than $\{p,q\}$-register (Sec. 3.3).

3.1 Failure detector σ

Intuitively, failure detector σ selects, for each run, a pair A of two processes (not necessarily correct), called the *active* processes: σ permanently outputs $\perp$ at all other processes. When all other processes are faulty, σ behaves like Σ_A at the active processes. That is, σ outputs at these two active processes, subsets of A that intersect and contain eventually only correct processes. In case the active processes are not the only correct ones in the system, σ might simply output $\emptyset$ at these active processes.

More precisely:

DEFINITION 3. *$\forall F \in \mathcal{E}$, $\forall H \in \sigma(F)$ there is a pair $A = \{p,q\}$ of processes verifying the following properties:*

- Well-formedness.
  - *$\forall t \in \Phi, \forall x \in A, H(x,t) \subseteq A$ and $\forall t \in \Phi, \forall x \notin A, H(x,t) = \perp$*
- Completeness.
  - *$\forall x \in A \cap Correct(F)$, $\exists t \in \Phi\ \forall t' > t\ \ H(x,t') \subseteq Correct(F)$*
- Intersection.
  - *$\forall x, y \in A, \forall t, t' \in \Phi$ if $H(x,t) \neq \emptyset$ and $H(y,t') \neq \emptyset$ then : $H(x,t) \cap H(y,t') \neq \emptyset$*
- Non-triviality.
  - *If $Correct(F) \subseteq A$ then : $\forall x \in A$, $\exists t \in \Phi$, $\forall t' > t\ H(x,t') \neq \emptyset$*

The algorithm described in Figure 2 implements set agreement using σ. The output of σ is obtained by the process using primitive $queryFD()$. In every run, exactly two processes are distinguished as active. Processes that are not active obtain $\perp$ and decide on their own value. On the other hand, active processes either decide a value coming from a non-active one or eliminate one of their initial values.

If σ outputs $\perp$ to a process, then this process does not belong to the set A of active processes. A non-active process sends to all its own value before deciding on it.

By the definition of σ, only two processes are in the set of active processes. Only active processes run Task 1 and Task 2. In Task 1, when it receives a value from a non-active process, an active process decides this value. Hence, an active process will be ensured to decide if at least one correct process is not active. If there is no correct process in the set of non-active processes, then the active processes will decide by Task 2. In Task 2, active processes reach consensus using σ.

THEOREM 4. *The algorithm of Figure 2 implements set agreement using σ.*

PROOF. Consider any run r with a failure pattern F. Let H be the associated failure detector history of σ. Let $A = \{q_0, q_1\}$ be the active set for H. v_p denotes the local variable of any distributed variable v of p (e.g., Me).

From the intersection property of σ we directly deduce the following fact:

```
Code for each process $p$:

1  to propose $(v)$ :
2    if $\perp = queryFD()$ then
3      send$(D, v)$ to all
4      decide$(v)$
5      return
6    else
7      start Task 1 and Task 2

8    Task 1:
9      upon receive$(D, *)$:
10       if $(D, w)$ has been received then
11         send$(D, w)$ to all
12         decide$(w)$
13         return

14   Task 2:
15       $Me \leftarrow v$; $You \leftarrow \perp$
16       Phase 1:
17           send $(1, Me)$ to every process except $p$
18           wait until received $(1, *)$ or $\{p\} = queryFD()$
19           if $(1, w)$ has been received then $You \leftarrow w$

20       Phase 2:
21           send $(2, You)$ to every process except $p$
22           wait until received $(2, *)$ or $\{p\} = queryFD()$
23           if $(2, \perp)$ has been received then $Me \leftarrow \perp$

24       Phase 3:
25                                  (* we assume that $\perp < v$ for all $v$ *)
26           $w \leftarrow \max\{Me, You\}$
27           decide$(w)$
28           return
```

Figure 2: Implementing set agreement using σ.

FACT 5. *For $i = 0$ or 1, if at some time t, process q_i gets $H(q_i, t) = \{q_i\}$ from σ, then at all times t', q_{1-i} gets $H(q_{1-i}, t') \neq \{q_{1-i}\}$.*

Termination: All correct non-active processes decide in Line 4. Let q be a correct process in the set of active processes. If at least one non-active process is correct then all correct active processes that do not decide by Task 2 will decide and terminate by Task 1, in Line 12. So, we only have to contradict the existence of a run in which (1) all non-active processes are faulty, and either (2) q_0 never decides or (3) q_1 never decides.

Assume the existence of such a run α with failure pattern F_α and failure detector history $H_\alpha(F_\alpha)$ for σ. By hypothesis, $Correct(F_\alpha) \subseteq \{q_0, q_1\}$. Because there is at least one correct process in each failure pattern then we can assume without loss of generality that q_0 is correct.

We have two cases to consider:

(a) q_1 is also a correct process. Both q_0 and q_1 send a message in Line 17 and then q_0 and q_1 terminate the repeat loop of Phase 1. Then both q_0 and q_1 send a message in Line 21 and then q_1 and q_0 terminate the repeat loop of Phase 2. Hence q_0 and q_1 decide in Lines 27 or 12 and terminate, contradicting the existence of the run α.

(b) q_1 is faulty, then q_0 is the only correct process and by the non-triviality and the completeness properties of σ there is a time after which $H_\alpha(q_0, *)$ equals $\{q_0\}$. Consequently q_0 is not blocked forever in the repeat loop of Task 2 and decides in Lines 27 or 12, contradicting the existence of the run α.

Validity: We have only to check that if an active process q decides by Line 27 then $w_q \neq \perp$. Without loss of generality, assume that q_0 decides $\perp$ in Line 27. For this to occur when q_0 decides we have $Me_{q_0} = You_{q_0} = \perp$.

- From $You_{q_0} = \perp$, we deduce that q_0 did not receive any message from q_1 in Line 19 and q_0 terminated the repeat loop of Phase 1 because $queryFD()$ outputs $\{q_0\}$.
- From $Me_{q_0} = \perp$, we deduce that q_0 received a message $(2, \perp)$ in Line 23. If q_1 sent $(2, \perp)$ then q_1 did not receive a message $(1, *)$ from q_0 in Line 19 and q_1 terminated the repeat loop of Phase 1 because $queryFD()$ outputs $\{q_1\}$.

These values of the ouputs of failure detector σ contradict Fact 5.

Agreement: Non-active processes decide at most $n - 2$ different values. If at least one active process decides by Task 1, then this process decides one of these $n - 2$ values and so there is at most $n - 1$ decided values. It remains to show that if both active processes decide by task 2 then they decide the same value.

By Fact 5, if both active processes decide, then at least one active process terminates its repeat loops by receiving message $(1, *)$ and message $(2, *)$. Assume without lost of generality that q_0 receives messages from q_1 in Line 19 and Line 23. So q_0 sends message $(1, v_{q_0})$ and $(2, v_{q_1})$. There are two cases to consider:

(a) q_1 receives message $(1, v_{q_0})$ from q_0 in Line 19. So when q_0 and q_1 decide in Line 27 we have: $Me_{q_0} = v_{q_0}$, $You_{q_0} = v_{q_1}$, $Me_{q_1} = v_{q_1}$ and $You_{q_1} = v_{q_0}$.

```
Code for $p_i$:
if  $p_i \in \{p,q\}$ then
    while  true do
        $Y \leftarrow queryFD()$
        if  $Y \subseteq \{p,q\}$ then
            $output \leftarrow Y$
        else
            $output \leftarrow \emptyset$
else
    $output \leftarrow \bot$
```

Figure 3: $\sigma \preceq \Sigma_{\{p,q\}}$

(b) q_1 does not receive message $(1, v_{q_0})$ from q_0 in Line 19. So when q_0 and q_1 decide in Line 27: $Me_{q_0} = \bot$, $You_{q_0} = v_{q_1}$, $Me_{q_1} = v_{q_1}$ and $You_{q_1} = \bot$.

In both cases, q_0 and q_1 decide on the same value. $\square$

3.2 A (2-)register is harder than set agreement

We now show that for any set $\{p,q\}$ a $\{p,q\}$-register is harder than set agreement. As the weakest failure detector for $\{p,q\}$-register is $\Sigma_{\{p,q\}}$[9], we simply prove here that $\sigma \preceq \Sigma_{\{p,q\}}$. The algorithm that implements σ using $\Sigma_{\{p,q\}}$ is given in Figure 3.

Lemma 6. *$\forall p,\ q \in \Pi$: $\sigma \preceq \Sigma_{\{p,q\}}$*

Proof. Let p and q be any two processes, the algorithm of Figure 3 emulates the output of σ using $\Sigma_{\{p,q\}}$. The emulated failure detector history is abstracted by the variable *output*. $output_p^t$ denotes the value of *output* of process p at time t.

We prove the properties of σ. The set of active processes is $\{p,q\}$. *output* is $\bot$ for processes different from p and q and is a subset of $\{p,q\}$ for process p or process q: this ensures the well formedness property. Completeness of $\Sigma_{\{p,q\}}$ ensures completeness of σ.

Consider a run r with failure pattern F and some failure detector history H of $\Sigma_{\{p,q\}}(F)$.

For the intersection property, consider $x, y \in \{p,q\}$, $t, t' \in \Phi$ with $output_x^t \neq \emptyset$ and $output_y^{t'} \neq \emptyset$. As $output_x^t$ and $output_y^{t'}$ are respectively equal to $H(x,u)$ and $H(y,u')$ for some $u, u' \in \Phi$ and $H(x,u) \cap H(y,u') \neq \emptyset$ we have $output_x^t \cap output_y^{t'} \neq \emptyset$ proving the intersection property of the emulated failure detector.

For the non-triviality, assume $Correct(F) \subseteq \{p,q\}$. By the completeness of $\Sigma_{\{p,q\}}$, the output of $\Sigma_{\{p,q\}}$ is eventually forever a subset of$\{p,q\}$. Then, by the algorithm, for $x \in \{p,q\}$, $output_x$ is eventually forever an output of $\Sigma_{\{p,q\}}$. As there is at least one correct process in F, either p or q is correct. By the intersection property of $\Sigma_{\{p,q\}}$, the output is not empty, proving the Non-triviality. $\square$

3.3 Set-agreement is not harder than a (2-)register

For the special case of $n = 2$, the register and set agreement abstractions are equivalent [9]. But set agreement is not harder than a 2-register as soon as $n \geq 3$.

If set agreement was harder than a 2-register, as σ implements set agreement, σ would be stronger than the weakest failure detector for a 2-register. We use again the fact that, for a set $\{p,q\}$ of processes, $\Sigma_{\{p,q\}}$ is the weakest failure detector to implement a $\{p,q\}$-register. We show that set agreement is not harder than $\{p,q\}$-register by proving that there is no algorithm that emulates $\Sigma_{\{p,q\}}$ from σ.

Lemma 7. *If $n \geq 3$, $\forall p,\ q \in \Pi$: $\Sigma_{\{p,q\}} \not\preceq \sigma$*

Proof. We prove this result by contradiction. Assume there is an algorithm that implements $\Sigma_{\{p,q\}}$ using σ. The emulated failure detector history is abstracted by the variable *output*. Variable $output_p^t$ denotes the value of *output* of p at time t. As $n \geq 3$, besides p and q, there is another process, say a. In the following, in all failure patterns we consider, all processes of $\Pi - \{a,p,q\}$ are crashed from the beginning.

Consider first failure pattern F, in which p and a are correct and q is crashed from the beginning. A possible failure detector history H for $\sigma(F)$ has $A = \{p,q\}$ as the set of active processes and, for all times t, $H(p,t) = \emptyset$ and $H(a,t) = \bot$. Consider a run r with this failure pattern and this failure detector history. By the completeness property of $\Sigma_{\{p,q\}}$, there is a time t at which $output_p^t \subseteq \{a,p\}$.

Consider now a failure pattern F' in which q is correct and both p and a crash right after time t. Then let H' be a failure detector history of $\sigma(F')$ in which (1) $A = \{p,q\}$ is the set of active processes, (2) for all times $t' \leq t$ $H'(p,t') = H'(q,t') = \emptyset$ and $H'(a,t') = \bot$ and, (3) for all time $t' > t$ $H'(q,t') = \{q\}$.

Consider a run r' with this failure detector history H' in which p and a take the same steps until time t, as in r, and q takes its first step at time $t+1$. Then for all times $t' \leq t$ the value of $output_p^{t'}$ is the same in r and r'. By the completeness property of $\Sigma_{\{p,q\}}$, there is a time t'' after which $output_q^{t''} \subseteq \{q\}$. Then $output_p^t \cap output_q^{t''} = \emptyset$, contradicting the intersection property of $\Sigma_{\{p,q\}}$. $\square$

4. A $2K$-REGISTER IS STRICTLY HARDER THAN $(N-K)$-SET AGREEMENT

We generalize the results of section 3 to the more general case of a $2k$-register and $(n-k)$-set agreement. In the following, X_{2k} denotes a set of $2k$ processes. We prove the following:

Theorem 8. *$\forall n \geq 3$, $\forall k$, $1 \leq k \leq n/2$, $\forall X_{2k}$, a X_{2k}-register is strictly harder than $(n-k)$-set agreement.*

To prove our theorem, we introduce a family of new failure detectors σ_k, as natural extensions of σ.[2] In fact σ is σ_2. Failure detector σ_k is defined for all k between 1 and n. Intuitively, failure detector σ_k selects, for each run, a subset A of k processes, called the *active* processes: σ_k permanently outputs $\bot$ at all other (non-active) processes. However, σ_k behaves like Σ_A at the active processes in runs where all other processes are faulty. That is, σ_k outputs at the active processes, subsets of A that intersect and contain eventually only correct processes.

In case the active processes are not the only correct ones in the system, σ_k might simply output $\emptyset$ at the active processes.

Then we show that failure detector σ_{2k} implements $(n-k)$-set agreement (Sec. 4.1). We also prove that $\Sigma_{X_{2k}}$ is stronger than σ_{2k}, proving that a X_{2k}-register is harder than

[2]Failure detector σ cannot implement $(n-2)$-set agreement.

$(n-k)$-set agreement (Sec. 4.2). Finally we prove that σ_{2k} is not stronger than $\Sigma_{X_{2k}}$, deducing therefore that $(n-k)$-set agreement is not harder that a X_{2k}-register (Sec. 4.3).

4.1 Failure detector σ_k

Roughly speaking, failure detector σ_k selects, for each run, a subset A of k processes, called the *active* processes. Failure detector σ_k behaves like Σ_A at the active processes when all other processes are faulty.

The case $n = 2k$ is special because all processes are then active. For this case we weaken a little bit the intuitive definition of σ_k.

More precisely, the set of active processes A is decomposed into two subsets: $\mathcal{A}$ and $\bar{\mathcal{A}}$. $\mathcal{A}$ is the subset of A composed of the $\lfloor k/2 \rfloor$ smallest elements of A and $\bar{\mathcal{A}} = A \setminus \mathcal{A}$. Failure detector σ_k may give no information to processes in A (in this case the output for the processes in A is $(\emptyset, A)$), but if there is no correct process in $\mathcal{A}$ or no correct process in $\bar{\mathcal{A}}$ then σ_k outputs eventually for processes of A, subsets of A that intersect and contain eventually only correct processes.

More precisely:

Definition 9. *$\forall F \in \mathcal{E}$, $\forall H \in \sigma(F)$ there is a subset A of k processes satisfying the following properties:*

- Well-formedness.
  - *$\forall t \in \Phi, \forall x \in A, H(x,t) = \emptyset$ or (X, A) with $X \subseteq A$ and $\forall t \in \Phi, \forall x \notin A, H(x,t) = \perp$*
- Completeness.
  - *$\forall x \in A \cap Correct(F)$, $\exists t \in \Phi\ \forall t' > t\ \ H(x,t') = \emptyset$ or $H(x,t') = (X, A)$ with $X \subseteq Correct(F)$*
- Intersection.
  - *$\forall x, y \in A, \forall t, t' \in \Phi$, if $\exists X \subseteq A$, $X \neq \emptyset$ and $Y \subseteq A$, $Y \neq \emptyset$ such that $H(x,t) = (X, A)$ and $H(y,t') = (Y, A)$, then : $X \cap Y \neq \emptyset$*
- Non-triviality.
  - *Let $\mathcal{A}$ be the set of the $\lfloor k/2 \rfloor$ smallest processes in A. $\bar{\mathcal{A}} = A \setminus \mathcal{A}$.*
    If $Correct(F) \subseteq \mathcal{A}$ or $Correct(F) \subseteq \bar{\mathcal{A}}$ then : $\forall x \in correct(F)$, $\exists t \in \Phi$, $\forall t' > t\ H(x,t') \neq \emptyset$ and $H(x,t') \neq (\emptyset, A)$

The algorithm described in Figure 4 implements $(n-k)$-set agreement using σ_{2k}. In the code of this algorithm the output of σ_{2k} is obtained by a process using primitive *queryFD*(). The output of σ_{2k} may be a pair, in which case the first component is obtained by *queryFD*().*trust* and the second one by *queryFD*().*active*.

Let A be the $2k$ elements set of active processes chosen by σ_{2k}. The processes in $\Pi \setminus A$ are non-active and decide their own proposed values. At most $n - 2k$ values are thus decided by these processes. The processes of A, the active processes, have to decide on a subset of at most k values. To ensure this, the set A is partitioned into two subsets of k elements: $\mathcal{A}$ contains the k processes of A with the smallest identities and $\bar{\mathcal{A}} = A \setminus \mathcal{A}$. The algorithm will essentially try to eliminate either values from $\mathcal{A}$ or values from $\bar{\mathcal{A}}$.

Basically, the processes of $\mathcal{A}$ send their own values and decide only on values from $\bar{\mathcal{A}}$. As soon as a process p receives

```
Initialization: $\forall j, 1 \le j \le 2k, T[j] = \perp$

Code for each process $p_i, 1 \le i \le n$:

to propose ($v_i$) :
  if $queryFD().active = \perp$ then
    send($D, v_i$) to all
    decide($v_i$)
    return
  else
    start Task 1 and Task 2 in parallel

  Task 1:
    upon receive($D, *$):
      if ($D, w$) has been received then
        send($D, w$) to all
        decide($w$)
        return

    upon receive($v, i$) for the first time:
        send($v, i$) to all
        $T[i] \leftarrow v$
        return

  Task 2:
    $A \leftarrow \emptyset$
    while $A = \emptyset$ do
      $A \leftarrow queryFD().active$
    $\mathcal{A}$ is the $k$ smallest elements of $A$
    $\bar{\mathcal{A}}$ is the $k$ greatest elements of $A$

    if $p_i \in \mathcal{A}$ then
        send($v_i$,$i$) to all
        repeat
          $X \leftarrow queryFD()$
          if $\exists x$ such that $p_x \in \bar{\mathcal{A}}$ and $T[x] \neq \perp$ then
            decide($T[x]$)
            send($D, T[x]$) to all
            return
        until($X.active \neq \emptyset \wedge X.trust \neq \emptyset \wedge (\bar{\mathcal{A}} \cap X.trust = \emptyset)$)

    else /* $p_i \in \bar{\mathcal{A}}$ */
        repeat
          $X \leftarrow queryFD()$
          if $\exists x$ such that $p_x \in \mathcal{A}$ and $T[x] \neq \perp$ then
            send($T[x]$,$i$) to all
            decide($T[x]$)
            send($D, T[x]$) to all
            return
        until($X.active \neq \emptyset \wedge X.trust \neq \emptyset \wedge (\mathcal{A} \cap X.trust = \emptyset)$)
```

Figure 4: Implementing $(n-k)$-set agreement with σ_{2k}.

```
Code for $p$:
if $p \in X$ then
  while true do
    $Y \leftarrow queryFD()$
    if $Y \subseteq X$ then
      $output \leftarrow (Y, X)$
    else
      $output \leftarrow \emptyset$
else
  $output \leftarrow \bot$
```

Figure 5: $\sigma_{|X|} \preceq \Sigma_X$

a message with a value of some process, p stores its value in a local array T_p. The processes of $\bar{\mathcal{A}}$ try to read in T_p values from processes in $\mathcal{A}$ and decide on them if such values are available. There is of course the possibility that all processes of $\mathcal{A}$ are faulty and never send their value. In this case, the processes of $\bar{\mathcal{A}}$ have to decide on their own value. But information given by σ_{2k} to the processes in $\bar{\mathcal{A}}$ is not strong enough to know whether all processes of $\mathcal{A}$ are faulty and have not yet decided.

However, notice that the intersection property of σ_{2k} ensures that, if some process p of $\bar{\mathcal{A}}$ has no information about the failures of processes in $\mathcal{A}$ at time t (i.e. $H(p,t).trust \neq \emptyset$ and $\mathcal{A} \cap H(p,t).trust = \emptyset$), then every process q in $\mathcal{A}$ has, at any time t', some information about the failures of processes in $\bar{\mathcal{A}}$ (i.e. $\bar{\mathcal{A}} \cap H(q,t').trust \neq \emptyset$ and $H(q,t').trust \neq \emptyset$). In the same way if some process p of $\mathcal{A}$ has no information about the failures of processes in $\bar{\mathcal{A}}$ at time t, then every process q in $\bar{\mathcal{A}}$ has, at any time t', some information about the failures of processes in $\mathcal{A}$.

Then after having sent its own value, a process in $\mathcal{A}$ will try, in a loop, to read and decide a value from the processes in $\bar{\mathcal{A}}$ until either it succeeds or its failure detector output has no information on failures of processes in $\bar{\mathcal{A}}$. Symmetrically, each process in $\bar{\mathcal{A}}$ will try in a loop to read and decide a value from processes in $\mathcal{A}$, until either it succeeds or its failure detector output has no information on failures of processes in $\mathcal{A}$.

If all processes end the loop by deciding, only values from $\mathcal{A}$ may be chosen for a decision and hence we have at most only k different values for the decision. The previous remark about the intersection property shows that if a process in $\mathcal{A}$ ends the loop without deciding then all processes in $\bar{\mathcal{A}}$ will choose a value from $\mathcal{A}$. Symmetrically if a process in $\bar{\mathcal{A}}$ ends the loop without deciding then all process in $\mathcal{A}$ will choose a value from $\mathcal{A}$. In this way k values are eliminated.

4.2 Σ_X is stronger than $\sigma_{|X|}$

We now show that, for any set X_{2k} of $2k$ elements, a X_{2k}-register is harder than $(n-k)$-set agreement. As the weakest failure detector for a X_{2k}-Register is $\Sigma_{X_{2k}}$[9], we simply show that $\sigma_{2k} \preceq \Sigma_{X_{2k}}$. The algorithm that implements σ_k using Σ_X for a subset X of k elements is given in Figure 5.

LEMMA 10. *$\forall X \subseteq \Pi$: $\sigma_{|X|} \preceq \Sigma_X$*

The proof is similar to the proof of Lemma 6. □

4.3 $(n-k)$-set agreement is not harder than a $2k$-register

If $(n-k)$-set agreement was harder than $2k$-Register, as σ_{2k} implements $(n-k)$-set agreement, then σ_{2k} would be stronger than the weakest failure detector for a $2k$-Register. We use again the fact that, for a set X_{2k} of processes, $\Sigma_{X_{2k}}$ is the weakest failure detector to implement a X_{2k}-Register. We show that $(n-k)$-set agreement is not harder than a X_{2k}-Register by proving that there is no algorithm that emulates $\Sigma_{X_{2k}}$ using σ_{2k}.

LEMMA 11. *If $n \geq 3$, $1 \leq 2k \leq n$, $\forall X_{2k} \subseteq \Pi$, $|X_{2k}| = 2k$: $\Sigma_{X_{2k}} \not\preceq \sigma_{2k}$*

PROOF. If $n \neq 2k$, the proof is similar to the proof of Lemma 7. In the special case $n = 2k$, we deduce the Lemma from the fact that if there a correct process in $\{p_1, \ldots, p_k\}$ and a correct process in $\{p_{k+1}, \ldots, p_n\}$ then the output of σ_n may be $(\emptyset, \Pi)$ and we cannot emulate a failure detector history of Σ from this. □

5. A $(2K+1)$-REGISTER IS NOT HARDER THAN $(N-(K+1))$-SET AGREEMENT

In this section, X_l denotes a set of l processes. The previous section showed that a X_{2k}-register is strictly harder that $(n-k)$ set agreement. We prove now that such a X_{2k}-register is not harder than $(n-k-1)$-set agreement.

Notice, in particular, that for a set X_2 of two processes, a X_2-register is strictly harder than set agreement but not harder than $(n-2)$-set agreement.

We first show the following:

THEOREM 12. *If $n > 2$, a register is not harder k-set agreement for $1 \leq k \leq \lceil n/2 \rceil - 1$*

PROOF. We proceed by contradiction. Assume there exists an algorithm $\mathcal{A}$ that implements $(\lceil n/2 \rceil - 1)$-set agreement using Σ. Consider now the environment where a majority of processes are correct. $\mathcal{A}$ implements $(\lceil n/2 \rceil - 1)$-set agreement using Σ in this environment too. But we can emulate Σ in this environment. Hence $\mathcal{A}$ implements $(\lceil n/2 \rceil - 1)$-set agreement in this environment. Therefore $\mathcal{A}$ implements $(\lceil n/2 \rceil - 1)$-set agreement in a shared memory distributed system with a majority of correct processes. This contradicts the results of [21, 13, 3] establishing the impossibility of k-set agreement if $k \leq (\lceil n/2 \rceil - 1)$. □

THEOREM 13. *For all sets X_{2k+1} of $2k+1$ processes with $1 \leq k \leq \lceil n/2 \rceil$ a X_{2k+1}-register is not harder than $(n-(k+1))$-set agreement.*

COROLLARY 14. *For all sets X_{2k} of $2k$ processes, a X_{2k}-register is not harder than $(n-(k+1))$-set agreement.*

PROOF. (Sketch) Let X be any set of $2k+1$ processes with $1 \leq k \leq \lceil n/2 \rceil$ and without loss of generality assume that $X = \{p_1, ..., p_{2k+1}\}$. We proceed by contradiction and assume there exists an algorithm $\mathcal{A}$ that implements $(n-(k+1))$-set agreement using Σ_X.

Consider algorithm $\mathcal{B}$ using failure detector Σ in a system of $2k+1$ processes with $\Pi = X$ such that each p_i executes in $\mathcal{B}$ exactly the same code as in $\mathcal{A}$.

With algorithm $\mathcal{A}$, each process p in $\Pi \setminus X$ has to decide even if all other processes have crashed from the beginning.

As p has no information about failures, for all failure pattern there is run in which p decides without receiving any message; in this case it decides its own value. By an easy induction, for all failure patterns there is run in $\mathcal{A}$ in which all processes in $\Pi \setminus X$ that decide in this run, decide their own value without receiving any message.

Then it is easy to verify that for each run of $\mathcal{B}$, there exists a run of $\mathcal{A}$ in which (1) p_i in X has the same behavior, (2) all processes in $\Pi \setminus X$ decide their own values and, (3) messages from processes in $\Pi \setminus X$ are delayed until the last decision of processes in X. As $\mathcal{A}$ implements $(n-(k+1))$-set agreement then the processes of X decide at most $n-k-1-(n-2k-1)) = k$ values. But then $\mathcal{B}$ using Σ implements k-set agreement contradicting Theorem 12. □

6. CONCLUDING REMARKS

Failure detectors for set agreement have been extensively investigated in the distributed computing literature, e.g. [20, 19, 17, 12, 18, 22, 7]. Most of these papers typically seek the weakest failure detector to implement set agreement in a shared memory model where registers are given as black-boxes. In a sense, the results of the present paper mean that determining the weakest failure detector to implement set agreement in a shared memory would not automatically yield the weakest in message passing. This contradicts for instance the conjecture of [22], that both the register and set agreement abstractions would require the same failure detector in a message passing system: we prove here that our new failure detector σ is powerful enough for set agreement but not sufficient for a register. In fact, we also prove in the appendix that $anti-\Omega$, the weakest candidate for set agreement in shared memory [22] does not implement set agreement in message passing.

The weakest failure detector to implement a register was determined in [8, 10]. The observation that even if a shared object O cannot implement an object O', the failure information needed to implement O might reveal sufficient to implement O' was first made in [9]. It was shown in [16] that a register shared by two processes is equivalent to one shared by any number of processes and this might seem to contradict our results. In fact, what is actually shown in [16] is that *any* number of registers shared by two processes can be used to implement a register shared by any number of processes. In our case, when we consider a k-register, we actually mean *one* register shared by a particular subset of k processes. Relations between set agreement and atomic objects were established in [11]. A partitioning approach was proposed in [7] to systematically weaken failure detectors for k-set agreement. At a high level, σ, when choosing a subset of active processes, makes indeed some kind of partition. However, σ is strictly weaker than the result of a partition applied to Σ.

7. REFERENCES

[1] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message passing systems. *J. ACM*, 42(2):124–142, Jan. 1995.

[2] H. Attiya and J. Welch. *Distributed Computing. Fundamentals, Simulations, and Advanced Topics.* McGraw-Hill, 1998.

[3] E. Boroswsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 91–100, May 1993.

[4] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, July 1996.

[5] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, Mar. 1996.

[6] S. Chaudhuri. More *choices* allow more *faults*: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, 1993.

[7] W. Chen, J. Zhang, Y. Chen, and X. Liu. Weakening failure detectors for k-set agreement via the approach. In *Proceedings of DISC*, volume 4731 of *Lecture Notes in Computer Science*, pages 123–138. Springer, 2007.

[8] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. Shared memory vs message passing. Technical Report 200377, EPFL Lausanne, 2003.

[9] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. (almost) all objects are universal in message passing systems. In *Proceedings of DISC*, volume 3724 of *Lecture Notes in Computer Science*, pages 184–198. Springer, 2005.

[10] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Koutnetzov, and S. Toueg. The weakest failure detectors to solve certain fundamental problems in distributed computing. In *Proceedings of the 23th ACM Symposium on Principles of Distributed Computing*, pages 338–346, July 2004.

[11] E. Gafni, M. Raynal, and C. Travers. Test & set, adaptive renaming and set agreement: a guided visit to asynchronous computability. In *Proceedings of SRDS*, pages 93–102, Oct. 2007.

[12] R. Guerraoui, M. Herlihy, P. Kouznetsov, N. A. Lynch, and C. C. Newport. On the weakest failure detector ever. In *Proceedings of the 26th ACM Symposium on Principles of Distributed Computing*, pages 235–243, Aug. 2007.

[13] M. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 111–120, May 1993.

[14] M. P. Herlihy. Wait-free synchronization. *ACM Trans. Prog. Lang. Syst.*, 13(1):123–149, Jan. 1991.

[15] L. Lamport. On interprocess communication; part I and II. *Distributed Computing*, 1(2):77–101, 1986.

[16] L. Lamport. On interprocess communication; part I: Basic formalism. *Distributed Computing*, 1(2):77–85, 1986.

[17] A. Mostéfaoui, S. Rajsbaum, and M. Raynal. The combined power of conditions and failure detectors to solve asynchronous set agreement. In *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, pages 179–188, July 2005.

[18] A. Mostéfaoui, S. Rajsbaum, M. Raynal, and C. Travers. Irreducibility and additivity of set agreement-oriented failure detector classes. In *Proceedings of the 25th ACM Symposium on Principles of Distributed Computing*, July 2006.

[19] A. Mostéfaoui and M. Raynal. k-set agreement with

limited accuracy failure detectors. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, pages 143–152, aug 2000.

[20] G. Neiger. Failure detectors and the wait-free hierarchy. In *Proceedings of the14th ACM Symposium on Principles of Distributed Computing*, pages 100–109, Aug. 1995.

[21] M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 101–110, May 1993.

[22] P. Zieliński. Anti-Ω: the weakest failure detector for set agreement. Technical report, UCAM-CL-TR-694, 2007.

Appendix

A. σ IS STRICTLY HARDER THAN $ANTI-\Omega$

Zielinski established in [22] that $anti - \Omega$ is the weakest failure detector to implement set agreement in an asynchronous shared memory system equipped with registers. Each query to $anti - \Omega$ returns a single process id; the specification ensures that there is a correct process whose id is returned only finitely many times. We show in this section that (1) $anti-\Omega$ is not sufficient to implement set agreement in an asynchronous message passing system (Sec. A.1) and (2) σ is strictly stronger than $anti - \Omega$ (Sec. A.2).

From (1) we deduce that $anti - \Omega$ is not the weakest failure detector to implement set agreement in an asynchronous message passing system. We also deduce from (2) that σ is not the weakest failure detector to implement set agreement in a shared memory equipped.

A.1 $anti-\Omega$ does not implement set agreement

LEMMA 15. *No algorithm implements set-agreement with $anti - \Omega$.*

PROOF. Assume there is an algorithm that implements set agreement using $anti - \Omega$.

Consider a failure pattern F_1 in which all processes are crashed from the beginning (at time t_0) except p_1. Let H_1 be any a failure detector history of $anti - \Omega(F)$. Let r_1 be any a run with this failure pattern and failure detector $H1$. By the termination property of set agreement, there is a time t_1 at which p_1 decides. By the integrity property of set agreement $p1$ decides its own proposed value.

Consider a failure pattern F_2 in which all processes are crashed from the beginning except p_2. Let H be any failure detector history of $anti - \Omega(F_2)$. Let H_2 be any failure detector history where for all $t \leq t_1$ $H_2(p,t) = H_1(p,t)$ and otherwise $H_2(x,t') = H(x,t')$. H_2 is a failure detector history of $anti - \Omega(F_2)$. Consider now a run r_2 with failure pattern F_2 and failure detector history H_2 such that p_2 takes its first step at time $t_1 + 1$. By the termination property of set agreement, there is a time $t_2 \geq t_1$ at which p_2 decides. As p_2 has not received any message from any other process, by integrity property of set agreement, p_2 decides its own proposed value.

We construct in this way a series of n failure patterns, failure detector histories and runs. Each time we can indeed extend the failure detector history because $anti - \Omega$ is an eventual failure detector.

Consider now a failure pattern F in which all processes are correct. H_n is a failure detector history of $anti - \Omega(F)$ for this failure pattern. Consider a run for failure pattern F and failure detector history H_n in which, for all i, (1) p_i takes its step between t_{i-1} and t_i as in r_i and takes no other step before t_{n+1}, (2) all messages sent by p_i are delayed after time t_{n+1}. This run is indistinguishable from r_i for p_i. Hence, at time t_i, p_i decides its own proposed value. Then all processes decide their own initial values contradicting the agreement property of $anti - \Omega$. $\square$

A.2 σ is strictly stronger than $anti - \Omega$

We first give an algorithm, depicted in Figure 6, that emulates $anti-\Omega$ using σ. The emulated failure detector history is abstracted by the variable *output*. All processes query their failure detector and know if they are in the active set

```
Code for $p_i$:
$nonactive \leftarrow \emptyset$; $active \leftarrow \emptyset$
start task 1 and 2
task 1:
    upon received $(NONACTIVE, p)$
        if $p \notin nonactive$ then
            send$(NONACTIVE, p)$ to all
            $nonactive \leftarrow nonactive \cup \{p\}$
    upon received $(ACTIVE, p)$
        if $p \notin active$ then
            send$(ACTIVE, p)$ to all
            $active \leftarrow active \cup \{p\}$
task 2:
    if $queryFD() = \bot$ then
            send$(NONACTIVE, p_i)$ to all
            $nonactive \leftarrow nonactive \cup \{p_i\}$
        else
            send$(ACTIVE, p_i)$ to all
            $active \leftarrow active \cup \{p_i\}$
    while $active \cup nonactive \neq \Pi$
        $output \leftarrow min\{p | p \notin active \cup nonactive\}$
    $min \leftarrow min\{p | p \in active\}$
    $max \leftarrow max\{p | p \in alive\}$          (* $active = \{min, max\}$ *)
    $output \leftarrow min$
    if $p_i = min$ then
        while $queryFD() \neq \{p\}$ do ;
        $output \leftarrow max$
        send$(CHANGE)$ to $max$
    else
        wait until received $(CHANGE)$
        $output \leftarrow max$
```

Figure 6: $anti - \Omega \preceq \sigma$

of σ or not. Then all processes broadcast a message indicating that they are alive and whether they are in the active set or in the non-active one. Then all processes collect the messages that have been sent in sets *active* and *nonactive*.

If a process p is crashed from the beginning, p will never be in $active \cup nonactive$ and p can be chosen as the value of the emulated failure detector history. If $active \cup nonactive$ contains all the processes, then all the processes know the set of active processes. In this case *output* will be one of the processes of the active set. Let p and q be the processes in the active set and assume $p < q$. To determine which process has to be output, we use again σ. There are two cases to consider: (A) p is faulty or p and some other process are correct and (B) p is the only correct process. In case (A), the *output* is p at every process: this gives a correct history of $anti - \Omega$. In case (B), the *output* must be changed. To detect this latter case, p queries σ. By the completeness and the non-triviality properties of σ , there is a time after which the query returns $\{p\}$. In this case p changes its output to q. Because it is possible that the query of σ returns $\{p\}$ to p even if q is correct, p indicates to q, by sending a message $CHANGE$, that it has changed its output. This is done to prevent the case where p outputs q and q outputs p when p and q are the only correct processes.

LEMMA 16. $anti - \Omega \preceq \sigma$

PROOF. The algorithm of Figure 6 uses σ to emulate $anti - \Omega$. We have to show that there is a correct process whose id is returned to all correct processes only finitely many times. Consider a failure pattern F, a failure detector history H of $\sigma(F)$ and a run r of the algorithm of Figure 6. Let A be the set of active processes given by H. Assume $A = \{p, q\}$ with $p < q$. We denote by $output^t_p$ the value of *output* of p at time t.

Note that the mechanism of sending and forwarding for $ACTIVE$ and $NONACTIVE$ implements a reliable broadcast: if a correct process receives such a message then all correct processes receive this message. And so eventually all correct processes have the same sets *active* and *nonactive*. There are two cases to consider:

1. $active \cup nonactive \neq \Pi$: Let x be the process with the minimal identity in $\Pi - active \cup nonactive$. As the communication is reliable, x is faulty. The choice of this process gives a correct failure detector history for $anti - \Omega$.
2. $active \cup nonactive = \Pi$ and p finds at some time t $H(p, t) = \{p\}$: In this case, for any time t, for all processes $x \in \Pi \setminus F(t)$: $output^t_x = p$ or $output^t_x = q$. If there is some correct process in *nonactive* then these values of $output_x$ constitute a correct failure detector history for $anti - \Omega$. If there is no correct process in *nonactive* then there are again three cases to consider:
    (a) p is the only correct processes: in this case by the completeness property of σ, there is a time t after which $H(p, t) = \{p\}$, and for any time $t' \geq t$, $output^{t'}_p = q$. These values of *output* constitute a correct failure detector history for $anti - \Omega$.
    (b) q is the only correct process: by the completeness property of σ, there is a time t after which $H(q, t) = \{q\}$, and so, by the intersection property, p (when it is alive) cannot get $H(p, t) = \{p\}$. Hence p never sends a $CHANGE$ message to q. Therefore, for any time $t' \geq t$, $output^{t'}_q = p$. These values of *output* constitute a correct failure detector history for $anti - \Omega$
    (c) p and q are correct: if p finds at some time t $H(p, t) = \{p\}$ then p and q change their output: $output^t_q = q$ and $output^t_p = q$ and never change after that. If p never finds at any time t $H(p, t) = \{p\}$, p and q never change their output: $output_q = p$ and $output_p = p$. In all cases, as p and q are correct, then these values of *output* constitute a correct failure detector history for $anti - \Omega$.

□

A Corollary of Lemma 15 is that the reverse is not true. And so σ is indeed strictly weaker than $anti - \Omega$.

COROLLARY 17. $\sigma \npreceq anti - \Omega$

PROOF. Assume that $\sigma \preceq anti - \Omega$. As σ implements set-agreement, $anti - \Omega$ implements set-agreement, contradicting Lemma 15. □