Renaming and the Weakest Family of Failure Detectors

Yehuda Afek — Tel-Aviv University
Petr Kuznetsov — TU Berlin/Deutsche Telekom Laboratories
Israel Nir — Tel-Aviv University

Abstract

We address the question of the weakest failure detector to circumvent the impossibility of $(2n-2)$-renaming in a system of up to n participating processes. We derive that in a restricted class of *eventual* failure detectors there does not exist a single weakest oracle, but a *weakest family* of oracles ζ_n: every two oracles in ζ_n are incomparable, and every oracle that allows for solving renaming provides at least as much information about failures as *one* of the oracles in ζ_n. As a by product, we obtain one more evidence that renaming is strictly easier to solve than set agreement.

Keywords: renaming, impossibility, synchrony assumptions, failure detectors

1 Introduction

Solving distributed computing problems in the presence of faults and in the absence of strong synchrony assumptions is a challenging task. It is often argued that these two factors are exactly what makes distributed computing research interesting. Indeed, most important problems are impossible to solve in purely *asynchronous* read-write shared-memory systems if processes participating in the computation may fail, even within the simple *crash* fault model in which a faulty process stops taking steps in the computation [6, 13, 22, 27]. Therefore, if we want to solve such problems in a fault-tolerant manner, we need to strengthen our synchrony assumptions.

A convenient approach to model partially synchronous systems is to use *failure detectors* [10,11]. Informally, a failure detector is a distributed service that provides hints about failures to processes participating in a distributed computation. At any moment of time, this information does not have to be accurate and complete, but it should satisfy certain axiomatic properties, depending on the specification of the failure detector. The notion of the *weakest* failure detector [10] captures the minimal information about failures that allows us to solve a given problem. That is, the information provided by the weakest failure detector for solving problem P is both sufficient and necessary to solve P.

In this paper, we focus on the weakest failure detector question in the context of *renaming* [2, 23]. In the problem of K-renaming, n processes come from a large set $\{p_1, \ldots, p_m\}$ $(m \geq 2n-1)$ of potential participants and choose new names in a smaller name space $1, \ldots, K$, so that no two processes choose the same name. We assume that the processes communicate by reading and writing in the shared memory. Typical values of K considered in this paper are $2n-1$ and $2n-2$.

Background. In the read-write shared memory model, the K-renaming problem has a *wait-free* solution when $K \geq 2n-1$ [5] (page 392, which is an adaptation of the message passing renaming algorithm of [2]).

Informally, a wait-free solution guarantees that each participating process obtains an output in a bounded number of its own steps, regardless of the processing delays or failures of other processes [20].

In [7], it was shown that for infinitely many values of n, $(2n-2)$-renaming is wait-free unsolvable. In this paper, we assume that n is such that $(2n-2)$-renaming is wait-free unsolvable, and we are interested in the minimal information about failures (encapsulated in a failure detector) that must be provided to make it solvable.

The problem of K-renaming is closely related to the (weak) *symmetry breaking* problem (SB) [18] (called *reduced renaming* in [23]). In the SB problem, up to n participating processes are required to output binary values. In every execution in which exactly n processes output, at least one process must output 0 and at least one process must output 1.

In fact, *weak* renaming ($K = 2n-2$, we simply call this case renaming in the following) is equivalent to SB [18]. In order to implement renaming from SB, we first observe that the $(2n-1)$-renaming algorithm of [2] has the property that if $k \leq n$ processes participate then they receive names in the range $\{1, \ldots, 2k-1\}$. Now SB can be used to partition the set of n participants into two non-empty sets S_0 (processes decided 0) and S_1 (processes decided 1). Then each class independently employs the wait-free algorithm of [2]. Processes in S_0 simply output the names returned by the algorithm of [2]. If a process in S_1 obtains name x from the algorithm of [2], then it outputs $2n-1-x$. If $|S_0| = r < n$, then the largest name that can be obtained by a process in S_0 is $2r-1$, and the smallest name that can be obtained by a process in S_1 is at least $2n-1-2(n-r)+1 = 2r$. Thus, every process obtains a unique name in the range $\{1, \ldots, 2n-2\}$. In the other direction, suppose that we can rename n processes in the range $\{1, \ldots, 2n-2\}$. Since there are exactly $n-1$ even names and exactly $n-1$ odd names in $\{1, \ldots, 2n-2\}$, taking the parity of the output name solves SB.

The requirement of at most n out of m processes participating in an execution can be formulated as a condition on the allowed *failure patterns*. Informally, a failure pattern in a given execution tells where and when failures might occur. In this paper we consider a specific set of failure patterns (an *environment* in the parlance of [10]) in which $m-n$ or more processes are initially faulty, and thus at most n processes can *participate*, i.e., take steps in the computation.

In this "n-participant" environment, denoted by $\mathcal{E}^n$, we are looking for the minimal information about failures required to solve renaming. Put differently, we are looking for the *weakest failure detector* to solve renaming in $\mathcal{E}^n$ [10]. Informally, $\mathcal{D}$ is the weakest failure detector for solving a given problem $\mathcal{M}$ if $\mathcal{D}$ is both (1) *sufficient* to solve $\mathcal{M}$, i.e., there exists an algorithm that solves $\mathcal{M}$ using $\mathcal{D}$, and (2) *necessary* to solve $\mathcal{M}$, i.e., any failure detector that is sufficient to solve $\mathcal{M}$ provides at least as much information about failures as $\mathcal{D}$ does. Renaming belongs to the large class of problems for which a weakest failure detector is guaranteed to exist [24].

But is this question meaningful? Strictly speaking, the weakest failure detector for solving the renaming problem in $\mathcal{E}^n$ is *renaming itself.* Indeed, consider a failure detector $\mathcal{D}_{ren}$ that outputs either $\perp$ or a name in the range $\{1, \ldots, 2n-2\}$. At every process, $\mathcal{D}_{ren}$ initially outputs $\perp$, but it may at some point switch to outputting a distinct value in $\{1, \ldots, 2n-2\}$. $\mathcal{D}_{ren}$ guarantees that no two processes are given the same name and that every correct process is eventually given a single distinct name.

$\mathcal{D}_{ren}$ is sufficient to solve renaming in $\mathcal{E}^n$: a process can simply wait until $\mathcal{D}_{ren}$ provides it with a (unique) name. $\mathcal{D}_{ren}$ is also necessary to solve renaming in $\mathcal{E}^n$, since it can be implemented given any algorithm that solves renaming in $\mathcal{E}^n$. Notice that this transformation of the renaming problem into the corresponding failure detector is possible due to the fact that renaming is an "input-less" task: the outputs depend only on the execution's schedule and not on the inputs of the participating processes.

Is this (trivial) solution satisfactory? Arguably, no: the "renaming" failure detector $\mathcal{D}_{ren}$ is obviously artificial, and to implement it in a given distributed system model is equivalent to solving renaming. Therefore, it does not provide any insights about the difficulty of solving renaming in "realistic" partially synchronous models.

In this paper we consider a class $\mathcal{Z}$ [29] of "realistic" failure detectors. This class is defined by three restrictions: First, we consider only *eventual* failure detectors, i.e., each failure detector is allowed to output any value in its range (the set of its possible outputs) for an unbounded but finite period of time. Second, we assume that the range of a failure detector is finite. Third, we assume that the "eventual" output of the failure detector depends solely on the set of faulty processes and not on the exact timing or order of failures. Most failure detectors in the literature are in $\mathcal{Z}$, including $\Diamond\mathcal{S}$ [11], Ω [10], anti-Ω [29] and $\neg\Omega_k$ [30]. Note that $\mathcal{D}_{ren}$ described above is not in $\mathcal{Z}$, since it is not eventual.

Contributions. In the class $\mathcal{Z}$ of failure detectors, satisfying the three conditions above (see also Section 4), we establish the existence of a *weakest family of failure detectors*: a collection ζ_n of mutually irreducible failure detectors (none of them can implement the other), such that each failure detector in $\mathcal{Z}$ that can solve a *non-trivial* task (i.e., a task that cannot be solved in the asynchronous wait-free model, specifically symmetry breaking), can implement *some* failure detector in ζ_n. Moreover, when $m = 2n - 1$, any element in ζ_n is strong enough to solve renaming. Given that every two distinct failure detectors in ζ_n are incomparable, we conclude that there is no unique weakest failure detector for renaming in $\mathcal{Z}$.

Note that this result does not contradict the previously mentioned result of Jayanti and Toueg [24] that many problems (including renaming) have a matching weakest failure detector. Indeed, given a collection C of failure detectors that solve a problem $\mathcal{M}$, the abstract reduction algorithm in [24] derives a failure detector that is weaker than any failure detector in C but still sufficient to solve $\mathcal{M}$. This constructed failure detector is however not eventual: based on the failures in any run the construction picks in an irrevocable manner an "algorithm" that solves $\mathcal{M}$. Therefore, it is not allowed to make mistakes. Naturally, $\mathcal{D}_{ren}$, the weakest failure detector for renaming, is not eventual either.

As a side product, this paper shows that the task of set consensus [12] is strictly harder than the task of renaming. In the environment $\mathcal{E}^n$, when $m = 2n - 1$, any element in ζ_n is strong enough to solve renaming. When $m > 2n - 1$, there exists a proper subset of detectors of ζ_n (and indeed, many such subsets) whose combined power solves renaming. However, in both cases, no proper subset of ζ_n can solve $(n-1)$-set consensus in $\mathcal{E}^n$.[1] This is the first evidence of the relationship between the two tasks derived for asynchronous systems equipped with failure detectors, complementing [17, 18].

Road map. Section 2 overviews the related work. Section 3 presents our model. Section 4 defines the class $\mathcal{Z}$ of failure detectors considered in this paper. Section 5 introduces ζ_n and overviews the results of this paper. Section 6 shows that no failure detector in ζ_n can be implemented by the other members of that family, and that every non-trivial failure detector implements at least on of the members in ζ_n. Section 7 explores the collective power of ζ_n to solve $(n-1)$-set consensus task. Section 8 shows that any element of ζ_n solves symmetry breaking and that ζ_n is a family of weakest failure detector to solve renaming. Section 9 compares elements of ζ_n to failure detectors in the set consensus hierarchy [15]. Section 10 discusses alternative perspectives on the problems considered in this paper and lists open questions.

[1]In the k-set consensus task (k-SC), the processes start with private inputs and produce outputs so that the set of output values is a subset of the set of inputs of size at most k. In $\mathcal{E}^n$, we say simply set consensus (SC) for $(n-1)$-set consensus.

2 Related Work

Attiya et al. [2] introduced the task of renaming and conjectured that the task is impossible to solve for n processes and an output name space of size $2n-2$ or less. Several papers later claimed to have proved this conjecture for all n, using instruments of algebraic topology [4,21,23]. Castañeda and Rajsbaum discovered that the claim is correct only for infinitely many values of n [8,9].

In this paper, we derive the minimal information about failures (expressed using the formalism of failure detectors) from the very impossibility of renaming and symmetry breaking. Therefore, the necessity part of our results hold for those values of n for which the tasks are impossible to solve [8]. Notice that for other values of n (namely, natural numbers the binomial coefficients of which are relative prime [8], the smallest such number is 6), the tasks of renaming and symmetry breaking can be solved wait-free [9].

The equivalence between renaming for an output name space of size $2n-2$ and SB was shown by Gafni et alii [18].

A stronger variant of the renaming problem called *adaptive* renaming was introduced by Attiya and Fouren [3]. Adaptive renaming additionally guarantees that if $1 \leq k \leq n$ participate, then all output names belong to the range $\{1, \ldots, 2k-2\}$. Gafni et al. [14, 16] showed that adaptive renaming is, in a strong sense, equivalent to $(n-1)$-set consensus: any algorithm solving one problem can be used for a "black-box" solution of the other. In this paper, we are focusing on (non-adaptive) renaming, for which the necessary and sufficient information about failures for this problem was not yet well understood.

Failure detectors, as oracles that describe synchrony assumptions to solve distributed computing problems, were introduced by Chandra and Toueg [11], and the first "weakest failure detector" was determined for the consensus task by Chandra, Hadzilacos, and Toueg [10]. Jayanti and Toueg showed that every problem in a large class of problems can be matched with the corresponding weakest failure detector [24].

Zieliński introduced the class $\mathcal{Z}$ of failure detectors and derived a complete classification of failure detectors in $\mathcal{Z}$ according their ability to solve distributed tasks [29]. In a subsequent paper, Zieliński showed that anti-Ω, the failure detector that outputs a process identifier so that some correct process is output only finitely many times, is the weakest failure detector for solving $(n-1)$-set consensus [30] in a system of $m = n$ processes. In [15], Gafni and Kuznetsov showed that $\neg\Omega_{n-1}$, a generalization of anti-Ω that outputs a set of $m-n+1$ processes such that some correct process is output only finitely many times, is the weakest failure detector for solving $(n-1)$-set agreement in a system of $m \geq n$ processes, in all environments, regardless of the assumptions on when and where failures might occur.

The current paper is based on [1], where a model of *loosely-named* systems was introduced. Informally, a loosely-named system assumes a set of n processes, where each process picks up an initial name in a large range of m names ($m \geq 2n-1$) before joining the computation. Respectively, the notion of loosely-named failure detectors is introduced in [1] and loosely-named failure detectors for set consensus, renaming and symmetry breaking, are described. Specifically, it is shown in [1] that *loose-anti-*Ω, a failure detector that outputs a process *name* so that eventually the name of some correct process is never output, is the weakest loosely-named failure detector in $\mathcal{Z}$ to solve the set consensus task.

This paper observes that, regarding the information about failures, the loosely-named systems can be replaced with a conventional system of m processes with a restriction that at most n of them are allowed to participate in the computation. The latter restriction can be naturally modeled as an *environment* $\mathcal{E}^n$ [10] which consists of failure patterns with at least $m-n$ initially faulty processes.

This simple observation allowed us to focus on the (most interesting) result of [1] concerning the weakest family of failure detectors. Indeed, loose-anti-Ω considered in $\mathcal{E}^n$ is equivalent to $\neg\Omega_{n-1}$ which is known to be the weakest failure detector for solving $(n-1)$-set consensus in any environment, including $\mathcal{E}^n$ [15].

3 Model

We consider a system of m processes $\Pi = \{p_1, \ldots, p_m\}$. Unless otherwise stated, throughout this paper we assume that the name of process p_i is i. Processes communicate by reading and writing in the shared memory and can query a failure detector [10, 11]. Processes are subject to *crash* failures: a failed process simply stops taking steps in the computation.

3.1 Failure patterns and failure detectors

A *failure pattern* F is a function from the time range $\mathbb{T} = \mathbb{N}$ to 2^{Π}, where $F(t)$ denotes the set of processes that have crashed by time t. Once a process crashes, it does not recover, i.e., $\forall t < t' : F(t) \subseteq F(t')$. We define $\mathit{faulty}(F) = \bigcup_{t \in \mathbb{T}} F(t)$, the set of faulty processes in F. Respectively, $\mathit{correct}(F) = \Pi \setminus \mathit{faulty}(F)$. When the failure pattern is well known, we use C to mark the set of correct processes. A process $p \in F(t)$ is said to be *crashed* at time t. An *environment* is a set of failure patterns. By default, we assume that at least one process is correct in every failure pattern.

A *failure detector history H with range $\mathcal{R}$* is a function from $\Pi \times \mathbb{T}$ to $\mathcal{R}$. $H(p_i, t)$ is interpreted as the value output by the failure detector module of process p_i at time t. A *failure detector $\mathcal{D}$* with range $\mathcal{R}_{\mathcal{D}}$ is a function that maps each failure pattern to a (non-empty) set of failure detector histories with range $\mathcal{R}_{\mathcal{D}}$. $\mathcal{D}(F)$ denotes the set of possible failure detector histories permitted by $\mathcal{D}$ for failure pattern F. We do not put any restriction on the possible ranges of failure detectors, apart from requiring them to be finite.

3.2 The n-participant environment, $\mathcal{E}^n$

Let $n \leq m$. In this paper, we restrict our attention to the *n-participant* environment $\mathcal{E}^n$ that consists of all failure patterns F such that $|F(0)| \geq m - n$. Notice that m is omitted from the notation. In other words, we only consider runs in which at most n processes are allowed to take steps. In case $m = n$, we say that the system is *tightly named*.

For a set $P \subseteq \Pi$, $|P| = n$, let environment $\mathcal{E}^P$ consist of failure patterns in which only processes in P *participate*, i.e., take steps: $\mathcal{E}^P = \{F \in \mathcal{E}^n \mid (\Pi \setminus P) \subseteq F(0)\} = \{F \in \mathcal{E}^n \mid \mathit{correct}(F) \subseteq P\}$.

In this paper, we assume $n \geq 2$ (for $n = 1$, the task of Renaming is trivial, and the task of SB is undefined). Note also that for $n = 2$ participants, our tasks of Renaming and SB are equivalent to consensus for which the weakest failure detector is known [10].

3.3 Algorithms

We define an *algorithm $\mathcal{A}$ using a failure detector $\mathcal{D}$* as a collection of deterministic automata, one automaton $\mathcal{A}_i$ for each process p_i. In each step of $\mathcal{A}_i$, process p_i performs an operation on a shared register, or queries its module of the failure detector $\mathcal{D}$ and receives a value, and then performs a state transition according to its automaton and the received values. A *step* is thus defined as a tuple that consists of a process id, the type of the step (*read*, *write*, or *query*), the accessed register if it is a memory (read or write) step, the read value if it is a read step, the written value if it is a write step, any process-local state transition and the failure detector value if it is a query step.

For each process p_i, the state of the automaton $\mathcal{A}_i$ includes a read-only input variable, denoted IN_i, and a write-once output variable, denoted OUT_i. In each initial state of $\mathcal{A}_i$, the input variable IN_i contains the input value of p_i and the output variable OUT_i is initialized to the special value $\perp$ (to denote that it has not yet been written by p_i).

3.4 Runs

A *state* of $\mathcal{A}$ defines the state of each process and each register in the system. An *initial state* I of $\mathcal{A}$ specifies the value of IN_i for every (participating) process p_i. A *run of algorithm $\mathcal{A}$ using a failure detector $\mathcal{D}$* in an environment $\mathcal{E}$ is a tuple $R = \langle F, H, I, S, T\rangle$ where $F \in \mathcal{E}$ is a failure pattern, $H \in \mathcal{D}(F)$ is a failure detector history, I is an initial state of $\mathcal{A}$, S is an *infinite* sequence of steps, respecting the automata $\mathcal{A}$ and the sequential specification of shared registers (each read step returns the value of the last write step on the same register), and T is an *infinite* list of increasing time values indicating when each step of S has occurred, such that for every query step $S[j]$ performed by process p_i that returned value d, $H(p_i, T[j]) = d$. We say that a run $R = \langle F, H, I, S, T\rangle$ is *fair* if every process in $\mathit{correct}(F)$ takes infinitely many steps in S.

3.5 Distributed tasks.

A *task* is defined through a set $\mathcal{I}$ of m-input vectors (one input value for each participating process that equals $\perp$ for non-participating processes) a set $\mathcal{O}$ of output m-vectors (one output value for each process that equals $\perp$ for non-terminated processes) and a total relation Δ that associates each input vector with a set of possible output vectors. Intuitively, in the task solution, processes start with inputs in $\mathcal{I}$ and eventually produce outputs in $\mathcal{O}$, so that the resulting input and output vectors are related by Δ.

In the *k-set consensus* task, each process takes a value in $\{0, \ldots, k\}$ as an input, and the set of non-$\perp$ output values is a subset of the input values of size at most k. In environment $\mathcal{E}^n$, the task of $(n-1)$-set consensus (we simply denote it by SC) is of special interest to us. A slightly stronger version of SC, called Strong Set Consensus [6], requires that if every participating process outputs, then at least one process outputs its own input value. It is straightforward to see that Strong Set Consensus is equivalent to SC [6]. Therefore, this paper assumes that every solution to SC satisfies this property.

3.6 Solving a task with a failure detector

Consider a task $\mathcal{T} = (\mathcal{I}, \mathcal{O}, \Delta)$ and an environment $\mathcal{E}$. We say that an algorithm $\mathcal{A}$ *solves $\mathcal{T}$ in $\mathcal{E}$ using a failure detector $\mathcal{D}$*, if in every fair run $\langle F, H, \mathit{IN}, S, T\rangle$ of $\mathcal{A}$, where $F \in \mathcal{E}$ and $\mathit{IN} \in \mathcal{I}$, every process in $\mathit{correct}(F)$ eventually *decides*, i.e., writes a non-$\perp$ output value to a write-once local variable OUT_p, so that the resulting vector OUT satisfies $(\mathit{IN}, \mathit{OUT}) \in \Delta$.

We write $\mathcal{D} \rightarrow_{\mathcal{E}^n} \mathcal{T}$ if there exists an algorithm, $\mathcal{A}$, that solves task $\mathcal{T}$ in environment $\mathcal{E}$ using a failure detector $\mathcal{D}$. If there is no such algorithm, we say that $\mathcal{D}$ doesn't allow solving $\mathcal{T}$ in $\mathcal{E}$, and use the notation $\mathcal{D} \not\rightarrow_{\mathcal{E}^n} \mathcal{T}$.

3.7 Comparing failure detectors

We say that failure detector $\mathcal{D}$ is *weaker* than failure detector $\mathcal{D}'$ in environment $\mathcal{E}$, and we write $\mathcal{D} \preceq_{\mathcal{E}} \mathcal{D}'$, if there exists an algorithm $\mathcal{A}$ using $\mathcal{D}'$ that emulates a *query* of $\mathcal{D}$ in $\mathcal{E}$. (We call $\mathcal{A}$ a *reduction* algorithm.) More precisely, $\mathcal{A}$ maintains, at every process p_i, a local variable $\mathcal{D}$-*output*$_i$ and guarantees that in every run with failure pattern F, there exists a history $H \in \mathcal{D}(F)$ such that for all $t \in \mathbb{T}$ and all process p_i, the value $\mathcal{D}$-*output*$_i$ at time t is $H(p_i, t)$. If $\mathcal{D} \preceq_{\mathcal{E}} \mathcal{D}'$ and $\mathcal{D}' \not\preceq_{\mathcal{E}} \mathcal{D}$, then we say that $\mathcal{D}$ is *strictly weaker* than $\mathcal{D}'$ in $\mathcal{E}$, and we write $\mathcal{D} \prec_{\mathcal{E}} \mathcal{D}'$.

$\mathcal{D}$ is the *weakest failure detector* to solve a task $\mathcal{M}$ in $\mathcal{E}$ if (i) there is an algorithm that solves $\mathcal{M}$ using $\mathcal{D}$ in $\mathcal{E}$ and (ii) $\mathcal{D}$ is weaker than any failure detector that can be used to solve $\mathcal{M}$ in $\mathcal{E}$. Every task can be shown to have a weakest failure detector [24].

A failure detector $\mathcal{D}$ is *trivial in $\mathcal{E}$* if there is an algorithm that emulates $\mathcal{D}$ in $\mathcal{E}$ without using the outputs of any other failure detector (we call such an algorithm *asynchronous*). For brevity, in such a case we may also say that $\mathcal{D}$ can be implemented in $\mathcal{E}$. A failure detector $\mathcal{D}$ is *non-trivial* if there is no such algorithm.

3.8 Anti-Ω and $\neg\Omega_k$

Two failure detectors are of special interest in this work, anti-Ω, introduced by Zieliński in [29] and its generalization $\neg\Omega_k$ introduced by Raynal [26].

Definition 1 *anti-Ω is a failure detector that outputs a process name whenever queried, such that there is at least one correct process whose name is returned only a finite number of times.*

In [30], Zieliński showed that anti-Ω is the weakest failure detector for solving $(m-1)$-set consensus in $\mathcal{E}^m$ (assuming that any set of processes can participate). This was followed [26] by the introduction of k-anti-Ω, or $\neg\Omega_k$ for short:

Definition 2 *$\neg\Omega_k$ is a failure detector that in every run outputs, when queried, a set of $m-k$ process names, such that at least one correct process is returned only a finite number of times.*

Trivially, $\neg\Omega_k$ is the same failure detector as anti-Ω when $k = m-1$, and thus $\neg\Omega_{m-1}$ can implement $(m-1)$-set consensus in $\mathcal{E}^m$. Moreover, when $k=1$, $\neg\Omega_k$ is equivalent to the Ω failure detector [10] which eventually returns the name of a single correct process (by taking the complement to $\neg\Omega_1$'s output), and thus, $\neg\Omega_1$ solves consensus.

In [30], Zieliński proved that for any k, $\neg\Omega_k$ can be used to implement k-set consensus in *any* environment, by showing that it is equivalent to another failure detector, $\overrightarrow{\Omega}_k$. In a nutshell, $\overrightarrow{\Omega}_k$ is a failure detector that outputs a vector of k entries, each is a process id. Eventually, at least one of the entries behaves like the Ω failure detector [10], that is, it stabilizes on the id of one of the correct processes. Finally, Gafni and Kuznetsov [15] showed that $\neg\Omega_k$ is the weakest failure detector for solving k-set agreement in any environment.

3.9 Renaming and symmetry breaking

The task of *K-renaming*, for the n-participant environment $\mathcal{E}^n$ $(m \geq n)$ (n out of m processes are allowed to participate), requires that every correct process eventually outputs a *name* in a *name space* $1,\dots,K$, and that no two processes that output a name choose the same name.

In the related task of *symmetry breaking* (SB), n processes have no input and output a binary value, 0 or 1. If all n processes output, at least one process outputs 0 and at least one process outputs 1. As we sketched in the introduction, SB is equivalent to $(2n-2)$-renaming. In [7], it was shown that for infinitely many values of n, SB and thus, $(2n-2)$-renaming (simply called Renaming in that paper), cannot be solved in the read-write asynchronous system.

Note that our environment-based definition of K-renaming slightly differs from the original definition of [2, 23] that requires n processes, $p_1,\dots,p_n$, each with an *input* in a large space $\{1,\dots,m\}$ to output names in a smaller space $\{1,\dots,K\}$ so that no two processes output the same name. To filter out trivial solutions that simply use process identifiers (e.g., every process p_i just outputs its identifier i), it is in [2, 23] assumed that the algorithm must be *anonymous*, i.e., may only depend on the inputs and not on the process identifiers.

It is easy to see that the two definitions of K-renaming, the input-based one and the environment-based one, are equivalent in the following sense: any read-write asynchronous algorithm to solve one version of K-renaming can be used to solve the other. Informally, an algorithm $\mathcal{A}^{inp}$ that solves input-based K-renaming for n processes can be used to solve the environment-based K-renaming in $\mathcal{E}$ by letting each process $p_i \in \{p_1, \ldots, p_m\}$ run the code of $\mathcal{A}^{inp}$ with input i. Similarly, an algorithm $\mathcal{A}^{env}$ that solves K-renaming in $\mathcal{E}^n$ can be used to solve the input-based K-renaming among n processes by letting each process with input i run the code of $\mathcal{A}^{env}$ corresponding to process p_i.

However, our version of K-renaming is easier to reason about in the models with failure detectors, since it does not put restrictions on the algorithms (it is hard to extend the notion of anonymous algorithms to failure-detector reductions). Note that in our model, K-renaming is an input-less task which allows us to easily determine the corresponding weakest failure detector, denoted $\mathcal{D}_{K\text{-}ren}$. The range of $\mathcal{D}_{K\text{-}ren}$ is $\{\perp, 1, \ldots, K\}$. Initially it outputs $\perp$ at every process. It may, eventually switch to outputting a distinct name in $\{1, \ldots, K\}$ so that (1) at every correct process, a name in $\{1, \ldots, K\}$ is eventually permanently output and (2) the same name is never output at different processes.

Obviously, $\mathcal{D}_{K\text{-}ren}$ allows for solving K-renaming in $\mathcal{E}^n$: every process simply waits until the output is provided by its failure detector module. Moreover, every algorithm solving K-renaming using any failure detector $\mathcal{D}$ can be used to implement $\mathcal{D}_{K\text{-}ren}$ in $\mathcal{E}^n$. Thus, $\mathcal{D}_{K\text{-}ren}$ is the weakest failure detector to solve K-renaming in $\mathcal{E}^n$. To eliminate such trivial solutions, in the next section, we put some natural restrictions on the scope of failure detectors we are willing to consider.

4 The class $\mathcal{Z}$ of failure detectors

In this paper, we make the following restrictions on the class of considered failure detectors [29].[2]:

Z1 A failure detector can behave arbitrarily for any finite amount of time.

Z2 The range of a failure detector is finite: $|\mathcal{R}_{\mathcal{D}}| < \infty$.

Z3 The output of a failure detector depends only on the set of correct processes and not on the timing of failures: $\forall F, F'$, $correct(F) = correct(F')$: $\mathcal{D}(F) = \mathcal{D}(F')$.

The class of failure detectors that satisfy **Z1** – **Z3** is denoted by $\mathcal{Z}$. We observe that the reasoning of [29] about the properties of failure detectors in $\mathcal{Z}$, originally derived for the "wait-free" environment $\mathcal{E}^m$, also works for any environment $\mathcal{E}^n$. Due to **Z3**, we may now define $\mathcal{D}(C)$ as the set of possible failure detector histories permitted by failure detector $\mathcal{D}$ when the set of correct processes is C.

Chiefly, a failure detector $\mathcal{D} \in \mathcal{Z}$ is unambiguously determined by the set of values that can be output infinitely often in any run using $\mathcal{D}$. Formally, let $inf(H)$, for a history H be the set of values appearing an infinite number of times in H. Let $infset_{\mathcal{D}}(C)$ denote the set of valid sets of values the failure detector $\mathcal{D}$ may return an infinite number of times when C is the set of correct processes, that is:

$$infset_{\mathcal{D}}(C) = \{inf(H) \mid H \in \mathcal{D}(C)\} \tag{1}$$

To define a failure detector in $\mathcal{Z}$ it is thus sufficient to define the value $infset_{\mathcal{D}}(C)$ for all $C \subseteq \Pi$. For instance, consider the $\neg\Omega_k$ failure detector that, as mentioned previously, outputs when queried a set of

[2]We slightly reformulate the properties of [29] to match our definitions. Specifically, property **Z3** is rewritten to conform with the standard failure detector formalism [10].

$m - k$ processes, so that eventually some correct process is never output. Formally:

$$infset_{\neg\Omega_k}(C) = \{T \subseteq 2^{\Pi} \mid (C \nsubseteq \cup_{S \in T} S \wedge (\forall S \in T. |S| = m - k)\} \qquad (2)$$

It is easy to see that the K-renaming failure detector $\mathcal{D}_{K\text{-}ren}$ defined in the previous section is not in $\mathcal{Z}$, since it is not eventual: it only allowed to switch from $\perp$ to a name in $\{1, \dots, K\}$ once.

5 The weakest failure detector family

Consider the environment $\mathcal{E}^n$. In the class $\mathcal{Z}$ of failure detectors we establish the existence of a *weakest family of failure detectors* in $\mathcal{E}^n$. We introduce a collection ζ_n of mutually irreducible non-trivial failure detectors, such that every non-trivial failure detector in $\mathcal{Z}$ can be used to implement *some* failure detector in ζ_n.

Definition 3 *Given a subset $S \subseteq \Pi$ of $n \geq 2$ processes (we simply say an n-*subset of Π *in the following), $\zeta(S)$ is a failure detector that returns a process name whenever queried. Eventually, $\zeta(S)$ returns only the names of processes in S. If some of the processes in S are correct, there is at least one correct process in S that $\zeta(S)$ returns only a finite number of times. Essentially, $\zeta(S)$ can be thought of as a localized version of anti-Ω for the subset S. Formally:*

$$infset_{\zeta(S)}(C) = \{T \mid (T \subseteq S) \wedge ((C \cap S \neq \emptyset) \Rightarrow (C \cap S \nsubseteq T))\}$$

Definition 4 *Let ζ_n be the set of $\binom{m}{n}$ different possible failure detectors $\zeta(S)$, one for each n-subset $S \subseteq \Pi$ in $\mathcal{E}^n$: $\zeta_n = \{\zeta(S) \mid S \subset \Pi, |S| = n\}$.*

Definition 5 *Let $\bar{\zeta_n}$ be a failure detector that returns a vector consisting of the outputs of the failure detectors in ζ_n in some arbitrary and well known order. That is, $\bar{\zeta_n}$ can be considered as the union of failure detectors in ζ_n.*

The failure detectors in ζ_n satisfy the following properties in environment $\mathcal{E}^n$, $n \geq 3$ (proofs are given in the next sections):

1. No failure detector in ζ_n can be implemented in $\mathcal{E}^n$, even in a system equipped with all the other failure detectors in the family. That is, $\forall \zeta \in \zeta_n,\ \zeta \npreceq_{\mathcal{E}^n} (\zeta_n \setminus \zeta)$ (Theorem 7 in Section 6).

2. Any non-trivial failure detector in $\mathcal{Z}$ implements at least one failure detector in ζ_n (Theorem 10 in Section 6).

3. $\bar{\zeta_n}$, the union of the failure detectors in ζ_n implements $(n-1)$-set consensus in $\mathcal{E}^n$, we simply write $\zeta_n \rightarrow_{\mathcal{E}^n}$ *SC* (Corollary 13 in Section 7).

4. No strict subset of ζ_n implements $(n-1)$-set consensus: $\forall \zeta \in \zeta_n,\ (\zeta_n \setminus \zeta) \nrightarrow_{\mathcal{E}^n}$ *SC* (Corollary 14 in Section 7).

5. When $m = 2n - 1$, there is an algorithm that solves Symmetry Breaking given the output of any failure detector in ζ_n: $\forall \zeta \in \zeta_n,\ \zeta \rightarrow_{\mathcal{E}^n}$ *SB* (Theorem 17 in Section 8).

6. When $m > 2n-1$, there is an algorithm that solves Symmetry Breaking in $\mathcal{E}^n$ using a subset of $O(\binom{\lceil m/2 \rceil}{n})$ failure detectors in ζ_n (Theorem 21 in Section 8).

Note that items 4, 5, and 6 above show that for $m \geq 2n-1$, solving Symmetry Breaking (and, thus, renaming) requires strictly less information about failures than solving set consensus. This complements the result of Gafni et al. [18] (which is extended in [17]) that in the iterated immediate snapshot model, renaming is weaker than set consensus to the asynchronous shared memory models with failure detectors.

6 The power of ζ_n

One can consider each $\zeta(S)$ as a localized version of anti-Ω, operating on a specific subset S. If there are no correct processes in S, eventually any output of $\zeta(S)$ is valid for anti-Ω. If, however, there is a correct process in S, $\zeta(S)$ returns the name of at least one such process only a finite number of times. This observation leads to Lemma 6, claiming that each $\zeta(S)$ is non-trivial, and to Theorem 10, claiming that the ζ_n family is the set of weakest failure detectors.

Lemma 6 *For any n-subset $S \subseteq \Pi$, $\zeta(S)$ cannot be implemented in $\mathcal{E}^S$.*

Proof. By contradiction, suppose that there exists an algorithm $\mathcal{A}$ that for some n-subset S implements $\zeta(S)$ in $\mathcal{E}^S$. We show that this implies that there exists an algorithm $\mathcal{A}'$ implementing anti-Ω [29] in a system Π' of n processes, which in turn means that SC is read-write solvable in Π'. Indeed, map every process $q_i \in \Pi'$ to a distinct process $\delta(q_i) \in S$ and let q_i run the algorithm of $\delta(q_i)$, implementing $\zeta(S)$. Eventually $\zeta(S)$ returns only names of processes in S (initially, whenever it returns a name of a process not in S, we may return the name of an arbitrary process in S as the output of anti-Ω). This would implement anti-Ω in Π' — a contradiction [29]. $\square$

Note that Lemma 6 implies that $\zeta(S)$ cannot be implemented in $\mathcal{E}^n$. Nevertheless, even though each $\zeta(S)$ is non-trivial in $\mathcal{E}^n$, it cannot implement $\zeta(S')$ for $S \neq S'$. In fact, no failure detector in ζ_n can be implemented by the collective power of all other failure detectors in ζ_n:

Theorem 7 *Given $S \subset \Pi$, $|S| = n$, $\zeta(S)$ cannot be implemented in $\mathcal{E}^n$ using $\zeta_n \setminus \{\zeta(S)\}$.*

Proof. It is enough to show that $\zeta(S)$ cannot be implemented by $\zeta_n \setminus \{\zeta(S)\}$ in the environment $\mathcal{E}^S$ that consists of all failure patterns in which only processes in S participate. Indeed, since $\mathcal{E}^S \subseteq \mathcal{E}^n$, this would imply that $\zeta(S)$ cannot be implemented from $\zeta_n \setminus \{\zeta(S)\}$ in $\mathcal{E}^n$.

First we show that for each subset of processes $S' \neq S$, $|S'| = n$, $\zeta(S')$ can be implemented in read/write in $\mathcal{E}^S$. We simply output any process $p \in S' \setminus S$. (Since $S' \neq S$ and $|S| = |S'|$, $S' \setminus S \neq \emptyset$.) Obviously, in $\mathcal{E}^S$, p is faulty, and thus is a valid output for $\zeta(S')$.

Therefore, for all $S' \neq S$, the failure detector $\zeta(S')$, and hence $\zeta_n \setminus \{\zeta(S)\}$, can be implemented in $\mathcal{E}^S$. Thus, $\zeta(S)$ cannot be implemented from $\zeta_n \setminus \{\zeta(S)\}$, otherwise we would obtain a read-write implementation of $\zeta(S)$ in $\mathcal{E}^S$, contradicting Lemma 6. $\square$

Our next step in showing the relative weakness of failure detectors in ζ_n is proving that any non trivial failure detector that belongs to the class $\mathcal{Z}$, implements at least one $\zeta(S)$. First, we show in the following lemma that in $\mathcal{E}^n$, for any non-trivial failure detector $\mathcal{D}$, there is at least one set of n participating processes for which $\mathcal{D}$ cannot be read/write implemented.

Lemma 8 *For any non-trivial failure detector $\mathcal{D} \in \mathcal{Z}$ in $\mathcal{E}^n$, there is an n-subset set $S \subseteq \Pi$, such that $\mathcal{D}$ is not implementable in $\mathcal{E}^S$.*

Proof. Assume to the contrary that it is possible to implement $\mathcal{D}$ for any participating set S of size n.

To reach a contradiction we show that $\mathcal{D}$ is trivial, i.e., it can be implemented using only read/write registers in any run in $\mathcal{E}^n$. Let each process record its participation upon waking up. When process p needs to query the output of $\mathcal{D}$, it scans the memory and checks the set of processes that have already recorded their participation. If p finds that n processes have already woken up, it can use the reduction algorithm that implements $\mathcal{D}$ in read/write (such an algorithm exists according to our assumption). If fewer than n processes have woken up, it chooses the first set (according to some predefined order known to all processes), $S \subseteq \Pi$, of size n, which contains all the processes that have already published their participation. It then implements $\mathcal{D}$ as though the participating set was S (and since $\mid S \mid= n$ it can do that in read/write).

To maintain consistency among the processes that implement $\mathcal{D}$ using the given reduction algorithm, processes should restart the reduction algorithm whenever they detect that a new process has woken up. More precisely, whenever process p needs to perform another step of the reduction algorithm, it first checks whether any new process has recorded its participation since it has previously scanned the set of participating processes, and if so, it restarts its algorithm. Note that if process p was the first to restart its algorithm after process q has recorded its participation, any other process p' will also restart its algorithm when it is to perform the next step in its algorithm.

Let t be a time after which no new process records its participation. Let $t' > t$ be the time after which no process restarts its reduction algorithm. Due to **Z1**, $\mathcal{D}$ may behave arbitrarily for any finite amount of time and thus the produced output is valid up to time t'. At time t', all the correct processes have already recorded their participation in the global array. The output of $\mathcal{D}$ is *simulated* (the corresponding $\mathcal{D}$-*output*$_i$ variable is maintained) by every correct process p_i using some set $S \supseteq C$ (C is the set of correct processes in that run). Since there is a run where the participating set is S, and the set of correct processes is C, the implementation of $\mathcal{D}$ using S is consistent with C (due to $\mathcal{D}$ satisfying **Z3**). Thus our produced output for $\mathcal{D}$ starting from time t' is valid, as the set of allowed histories of $\mathcal{D}$ is only dependent on C. Therefore, we obtain read/write implementation of $\mathcal{D}$, implying that $\mathcal{D}$—a contradiction. $\square$

Note that Lemma 8 holds even if $\mathcal{D}$ does not satisfy **Z2** (has an infinite range).

Lemma 9 *Let $\mathcal{D}$ be a failure detector in $\mathcal{Z}$. If there exists a subset of process $S \subset \Pi$, $|S| = n$, such that $\mathcal{D}$ is not read/write implementable when the participating set is S then $\zeta(S)$ is weaker than $\mathcal{D}$ in $\mathcal{E}^n$.*

Proof. We present an implementation of $\zeta(S)$ from $\mathcal{D}$. Let each process record its participation as the first step when it wakes up.

To query $\zeta(S)$ a process p first scans the set of participating processes. If there is a process $q \in S$ that has not recorded its participation, $\zeta(S)$ simply returns the name of q. If q never records its participation it follows that q is not a correct process, and in that case it is a correct implementation of $\zeta(S)$.

If, however, p finds that all the processes in S have woken up, since $|S| = n$, the participating set is S. In this case, $\mathcal{D}$ cannot be implemented in read/write due to our choice of S. $\mathcal{D}$ can then be considered as a non-trivial failure detector in a system of n processes. It follows from Theorem 12 of [29] that $\mathcal{D}$ can implement anti-Ω in such a system.

Thus, we have an implementation of anti-Ω for the processes of S, which when queried, returns only names of processes in S, and at least one of the correct processes in S is returned only a finite number of times (since S is the participating set, at least one of the processes in S is correct). We have thus implemented $\zeta(S)$. $\square$

Theorem 10 *Every non-trivial failure detector $\mathcal{D} \in \mathcal{Z}$ implements at least one of ζ_n's failure detectors in $\mathcal{E}^n$.*

Proof. By Lemma 8, there is a set S, such that when S is the participating set (that is, in the environment $\mathcal{E}^S$), failure detector $\mathcal{D}$ cannot be implemented in read/write. From Lemma 9, it follows that $\mathcal{D}$ implements $\zeta(S)$. $\square$

Theorems 7 and 10 imply that ζ_n is indeed the weakest *family* of non-trivial failure detectors in $\mathcal{Z}$: every non-trivial failure detector in $\mathcal{Z}$ can be used to implement at least one $\zeta(S) \in \zeta_n$ but no $\zeta(S)$ can be implemented using the collective power of $\zeta_n \setminus \zeta(S)$. As a result:

Corollary 11 *There does not exist a weakest non-trivial failure detector in $\mathcal{Z}$ in $\mathcal{E}^n$.*

7 Solving Set Consensus

We now show that $\bar{\zeta_n}$, the union of failure detectors in ζ_n solves the $(n-1)$-set consensus task.

Theorem 12 *In $\mathcal{E}^n$, $\bar{\zeta_n}$ and $\neg\Omega_{n-1}$ are equivalent, i.e., $\bar{\zeta_n} \preceq_{\mathcal{E}^n} \neg\Omega_{n-1}$ and $\neg\Omega_{n-1} \preceq_{\mathcal{E}^n} \bar{\zeta_n}$*

Proof. $\bar{\zeta_n}$ can implement $\neg\Omega_{n-1}$ in $\mathcal{E}^n$ as follows. Initially, every process records its participation. As long as the current set S of participating processes is of size less than n, every process outputs any set of $m-n+1$ processes not in S. When exactly n processes register their participation, every process outputs the union of $\Pi \setminus S$ and the current output of $\zeta(S)$ (which by itself is a process). Eventually, $\zeta(S)$ outputs a process in S that is not the only correct process in S. Therefore, the resulting unions are sets of $m-n+1$ processes, which eventually, never contain some correct process. Thus, $\neg\Omega_{n-1}$ is implemented.

Similarly, $\neg\Omega_{n-1}$ implements $\zeta(S)$ for every $S \subseteq \Pi$, $|S| = n$, in $\mathcal{E}^n$ (which implies in turn that $\neg\Omega_{n-1}$ implements $\bar{\zeta_n}$) as follows. As long as the currently observed set of participants P is not S, we output any process in $S \setminus P$ (there exists such a process since $|P| \leq n$ and $P \neq S$). If $P = S$, then we output any process in $S \cap S'$ where S' is the current output of $\neg\Omega_{n-1}$. Since $\neg\Omega_{n-1}$ outputs sets of $m-n+1$ processes that eventually never contain some correct process, $p_i \in C \subseteq P = S$, $S \cap S'$ is non-empty and eventually never contains p_i. Thus, $\neg\Omega_{n-1}$ implements $\bar{\zeta_n}$ in $\mathcal{E}^n$. $\square$

It is shown in [15] that $\neg\Omega_{n-1}$ is the weakest failure detector for solving $(n-1)$-set consensus in a system of $m \geq n$ processes, in all environments, including $\mathcal{E}^n$. Combined with Theorem 12, this implies that:

Corollary 13 *$\bar{\zeta_n}$ is the weakest failure detector for solving the $(n-1)$-set consensus task in $\mathcal{E}^n$.*

Since $\bar{\zeta_n} \in \mathcal{Z}$, Theorems 7, 12 imply:

Corollary 14 *No proper subset of ζ_n can solve $(n-1)$-set consensus in $\mathcal{E}^n$.*

In the following section we show how to solve symmetry breaking (SB) using proper subsets of ζ_n, which implies that SB does not have a single weakest failure detector in $\mathcal{Z}$.

8 Symmetry Breaking Implementations

First, we present a simple algorithm using a single $\zeta(S)$ that solves Symmetry Breaking (SB) in $\mathcal{E}^n$ when $m = 2n - 1$, and then a more elaborate algorithm solving SB when $m > 2n - 1$ using the combined power of several detectors in ζ_n.

8.1 Symmetry Breaking implementation for $m = 2n - 1$

Algorithm 1 provides a simple implementation of the SB task using a single, arbitrary $\zeta(S)$ for the case $m = 2n - 1$ and $n \geq 3$.

Each process $p \notin S$ in Algorithm 1 outputs 0 in Line 3. On the other hand, processes that are in S run a Strong $(n-1)$-set consensus task among themselves by using the algorithm given in [30] for $m = n$ and $\zeta(S)$ as an implementation of anti-Ω for the processes in S. Processes that decide on their own name, output 1 in Line 6 while processes that decide on a name different than their own, output 0 in Line 8. Since $m = 2n - 1$, if all n processes output, there must be at least one participating process in S, and, thus, at least one process outputs 1.

We are able to run the $(n-1)$-set consensus algorithm of [30], since the processes in S only interact with each other, and are not affected by processes outside of S, essentially, making them a closed system of n processes. In that system, $\zeta(S)$'s output can be used in lieu of anti-Ω's output. The reader should note that this construction solves the $(n-1)$-set consensus task only for the processes in S and thus is not a contradiction to Corollary 13.

Lemma 15 *(wait freedom) A correct process in Algorithm 1 eventually writes an output and terminates.*

Proof. The only way process p_i can get stuck in the algorithm is while executing the SC algorithm in Line 4. Since the SC algorithm described in [30] terminates after a finite number of steps once the anti-Ω failure detector stops outputting the name of one of the correct processes, it may get stuck only if our simulation of anti-Ω using $\zeta(S)$ is invalid. That is, none of the correct processes in S are returned only a finite number of times by $\zeta(S)$ (if there are no correct processes in S, processes executing the SC algorithm cannot get stuck since they eventually fail). However, by definition $\zeta(S)$ returns one of the correct processes $q \in S$ only a finite number of times, and due to the condition in Line 2, this process participates in the SC task as well. Therefore, q is considered correct by the process participating in the SC task and is returned only a finite number of times by our simulation of anti-Ω, as required. $\square$

Lemma 16 *(safety) When n processes reach a decision in Algorithm 1, at least one of them outputs* 0 *and at least one of them outputs* 1.

Proof. Let P be the participating set in a given run. Since n processes reach a decision, $|P| = n$. Therefore, we may consider two possible cases

(case 1) $P \neq S$. There is at least one process, $p \in P$, such that $p \notin S$. Process p finds the condition in Line 2 correct, and therefore outputs 0. On the other hand, since $|S| = n$ and $m = 2n - 1$, there is at least one process, $q \in P$, such that $q \in S$. Process q finds the condition in Line 1 false, and therefore participates in the SC task of Line 4. Among those processes participating in the SC task, at least one decides on its own name, and thus, finds the condition in Line 5 correct and outputs 1 in the following line. Therefore, at least one process outputs 0 and one outputs 1, as required.

Algorithm 1: Solving SB algorithm using $\zeta(S)$ for $m = 2n - 1$.

```
Output: Either 0 or 1
1 Process $p_i$'s Break Symmetry Procedure
2     if ($p_i \notin S$) then
          // process does not belong to the group on which the failure
             detector is defined
3         return 0
4     $dec$ := SC.propose($i$) // Run an anti-$\Omega$ based Strong $(n-1)$-SC algorithm
          among the processes in $S$.  Simulate anti-$\Omega$ using the output
          of $\zeta(S)$.
5     if ($dec = i$) then
6         return 1
7     else //  $dec \neq i$
8         return 0
```

(case 2) $P = S$. Since all the processes of S are participating, n processes find the condition in Line 2 false, and thus, n processes execute the set consensus task. Therefore, at least one process decides upon its own name, and outputs 1 in Line 6, and at least one process does not decide upon its own name, and outputs 0 in Line 8, as required. □

From the previous two lemmas we have:

Theorem 17 *Algorithm 1 solves SB in $\mathcal{E}^n$ using a single $\zeta(S)$ for $m = 2n - 1$.*

8.2 Symmetry Breaking implementation for $m > 2n - 1$

When we consider the case $m > (2n-1)$, a single $\zeta(S)$ is far from sufficient in order to solve the Symmetry Breaking task. The reader should note that one cannot trivially use a renaming algorithm in order to simulate a system of $2n - 1$ processes and utilize Algorithm 1 in order to solve SB. This is due to the fact that a failure detector cannot adopt the new naming scheme, since failure detectors' outputs may not be affected by processes' actions. Next, we present an upper bound on the number of ζ's required to solve SB in such systems. Let us first define the following failure detector:

Definition 18 *In an $\mathcal{E}^n$ environment where $m \geq 2n - 1$, consider the subsets $B = \{p_1, \ldots, p_{\lceil m/2 \rceil}\}$ and $W = \{p_{\lceil m/2 \rceil + 1}, \ldots, p_m\}$. Let ζ-*Set *be a failure detector that returns a vector consisting of the outputs of each $\zeta(S)$ such that S is a subset of either W or B and $|S| = n$. More formally, ζ-Set returns the output of each $\zeta(S) \in \{\zeta(S) \mid (S \subset B \vee S \subset W) \wedge |S| = n\}$.*

It is easy to see that each response of ζ-Set is composed from the outputs of $\binom{\lceil m/2 \rceil}{n} + \binom{\lfloor m/2 \rfloor}{n}$ failure detectors. Algorithm 2 is an implementation of the SB task for $m \geq 2n - 1$ processes (out of which at most n processes participate) using ζ-Set. At the heart of the algorithm is a division of the processes into two groups, $\{p_1, \ldots, p_{\lceil m/2 \rceil}\}$ and $\{p_{\lceil m/2 \rceil + 1}, \ldots, p_m\}$; Processes that belong to the first group set their "default" output value to 0, while others set it to 1 (Lines 6–9).

Each process p_i has two shared SWMR registers, $Output[i]$ and $Participating[i]$. $Output[.]$ records the decisions reached by the processes. In $Participating[.]$ each process records its participation as soon as it wakes up (Line 2), and it then scans $Participating[.]$ in order to acquire V the set of currently participating processes (Line 3). Next, p_i checks if either fewer than n processes are participating or if $\zeta(V)$ is not one of the ζ's provided by ζ-Set (Line 10). If this is the case, p_i outputs its default value. If p_i detects the participation of n processes, such that $\zeta(V)$ is one of the outputs provided by ζ-Set, it executes a Strong set consensus algorithm based on anti-Ω using $\zeta(V)$, as discussed previously. If p_i decides on its own name (Line 16) it returns the complement to its default value. If it decides on another process name (Line 18), it returns its default value. Concurrently while running the set consensus algorithm, p_i checks $Output$. If p_i detects that some other process has recorded an output (Line 20), it returns the complement of that output. The reason for doing so, is to avoid an edge-case where the correct processes names that are returned by $\zeta(V)$ only a finite number of times all belong to processes that do not participate in the set consensus task, invalidating the algorithm. Such processes can avoid participating in the SC task only by finding the condition in Line 10 true, followed by writing to their $Output$ register.

Lemma 19 *(wait freedom) A correct process in Algorithm 2 eventually writes an output and terminates.*

Proof. The only way process p_i can get stuck in the algorithm is in the loop of Lines 13–15, when the set consensus algorithm does not terminate and no other process records a decision in the $Output$ array. Since the SC algorithm described in [30] terminates after a finite number of steps once the anti-Ω failure detector stops outputting the name of one of the correct processes, it may get stuck, only if our simulation of anti-Ω using $\zeta(V)$ is invalid.

In the context of Algorithm 2, the simulation of anti-Ω may fail when a correct process, p_j, that was returned only a finite number of times by $\zeta(V)$, is not taking part in the set consensus algorithm (since it found the condition in Line 10 true). As a result, for the set consensus algorithm, p_j seems to be faulty. However, in such a case, since p_j is correct, it would eventually write its output (Line 11). Therefore, p_i would eventually see p_j's output (Line 15), and would exit the loop. $\square$

Lemma 20 *(safety) When n processes reach a decision in Algorithm 2, at least one of them outputs* 0 *and at least one of them outputs* 1.

Proof. Let P be the participating set. Due to our assumption that n processes reach a decision, $|P| = n$. If any process outputs the complement of another process's output, the validity of the proposition is trivial. Therefore, assume that no process determines its output by taking the complement of another process's output. Let r be $\lceil m/2 \rceil$ (as it is defined in the algorithm). We consider the following two cases:

(case 1) $P \nsubseteq \{p_1, \ldots, p_r\}$ and $P \nsubseteq \{p_{r+1}, \ldots, p_m\}$. In this case, every participating process finds the condition in Line 10 true, and thus outputs its default value (set in Lines 7 and 9). Since some processes are in $\{p_1, \ldots, p_r\}$ while some others are in $\{p_{r+1}, \ldots, p_m\}$, at least one process has a default output value 0, and at least one other process has the default output value 1 (Lines 6–9). Therefore, at least one process outputs 0 and at least one outputs 1.

(case 2) $P \subseteq \{p_1, \ldots, p_r\}$ or $P \subseteq \{p_{r+1}, \ldots, p_m\}$. Without loss of generality, assume that $P \subseteq \{p_1, \ldots, p_r\}$. Since at least one process detects the participation of all n processes in Line 3, at least one finds the condition in Line 10 false, and thus at least one participates in the set consensus algorithm. Since all processes that participate in the set consensus algorithm reach a decision (and, due to the assumption, do not terminate the algorithm prematurely by taking the complement of another process's decision), at least one process, p_i, decides on its own name. Thus, p_i outputs the complement of its default value (1). If n processes participate in the set consensus algorithm, at least one of them does not decide on its own name,

Algorithm 2: Solving SB with ζ-Set

```
Shared Variables:
    $Participating[1 \ldots m]$ : registers initialized with 0.
    $Output[1 \ldots m]$ : registers initialized with $\perp$
Local Variables:
    $v$, $V$, $r$ : local variables for each process
Process $p_i$'s Break Symmetry Procedure
    $Participating[i] \leftarrow 1$
    $V \leftarrow \{k | Participating[k] = 1\}$ // the set of observed participating processes
    $r \leftarrow \lceil m/2 \rceil$
    $dec \leftarrow \perp$ // local variable for each process
    if ($i \in \{1, \ldots, r\}$) then
        $v \leftarrow 0$
    else
        $v \leftarrow 1$

    if ($|V| < n$) or ($V \nsubseteq \{1, \ldots, r\}$ and $V \nsubseteq \{r+1, \ldots, m\}$) then
        $Output[i] \leftarrow v$, return $v$
    else // $n$ processes are participating, $\zeta(V)$ is given by $\zeta$-Set
        repeat
            Perform a single step in an anti-$\Omega$ based Strong $(n-1)$-SC algorithm among the
            processes in $V$, where each process proposes its own name. Simulate anti-$\Omega$ by using the
            output of $\zeta(V)$. Let $dec$ hold the decision, if one is reached
        until ($dec \neq \perp$) or ($\exists k. Output[k] \neq \perp$)

    if ($dec = i$) then
        // $p_i$ decided on its own name
        $Output[i] \leftarrow 1 - v$, return $1 - v$
    else if ($dec \neq \perp$) then // $p_i$ decided on another process name
        $Output[i] \leftarrow v$, return $v$
    else if ($\exists k. Output[k] \neq \perp$) then
        $Output[i] \leftarrow 1 - Output[k]$
        return $Output[i]$
```

and thus, at least one of them outputs its default value (0). Otherwise, if there exists a process p_k that does not participate in the set consensus algorithm, it surely finds the condition in Line 10 false, and thus outputs its default value (0). Therefore, at least one process outputs 0 and at least one other process outputs 1, as required. Similar considerations apply to the case when $P \subseteq \{p_{r+1}, \ldots, p_m\}$. □

The previous two lemmas lead to:

Theorem 21 *Algorithm 2 solves the Symmetry Breaking task in $\mathcal{E}^n$, $m \geq 2n-1$, using a ζ-Set failure detector.*

A natural lower bound on the number of ζ's necessary to solve SB in $\mathcal{E}^n$ can be reached by reducing the problem back to the case of $m = 2n - 1$. For example, when $m = 2n$, if the SB task was implementable using a single $\zeta(S)$, one could solve the SB task for $2n - 1$ processes without any failure detector. Assume that there is an algorithm $\mathcal{A}$ that solves SB for $m = 2n$, using a single arbitrary $\zeta(S)$. In order to solve SB for $2n - 1$ processes, one could map the process names from $\{1, \ldots, 2n-1\}$ to $\{1, \ldots, 2n\}$ in such a way that no two process names are mapped to the same name, and there exists a name $p \in S$ to which no process is mapped to (since we are left with $2n - 1$ names after the later restriction, such a mapping is possible). Using this mapping, one can simulate running $\mathcal{A}$ in a system of $2n$ processes. Whenever $\zeta(S)$ is queried in $\mathcal{A}$, its output can be simulated by taking the name of p as its output. This is a valid output for $\zeta(S)$, since p is not a correct process in the simulation. Therefore, as $\mathcal{A}$ solves SB for $2n$ processes, it solves SB in our simulation of such a system, and the processes can adopt the resulting outputs.

The following theorem is a generalization of the arguments given above. It presents a necessary condition on the set of ζ failure detectors that should be available in order to solve the SB task in $\mathcal{E}^n$.

Theorem 22 *Consider a set of failure detectors $\zeta_F = \{\zeta(S_1), \ldots, \zeta(S_k)\}$, and a system with $m \geq (2n-1)$ processes such that SB cannot be solved wait-free in a system of $2n - 1$ processes [7]. If there is a subset of $(2n-1)$ processes $G = \{q_1, q_2, \ldots, q_{2n-1}\}$ such that for every subset $H \subset G$, $|H| = n$, it holds that $\zeta(H) \notin \zeta_F$, then SB cannot be solved in $\mathcal{E}^n$ using ζ_F, when $m \geq (2n-1)$.*

Proof. Assume to the contrary that one can solve SB in $\mathcal{E}^n$ equipped with ζ_F as defined in the Theorem's definition by using algorithm $\mathcal{A}$. We show that if this is the case, one could read/write solve SB for $m = 2n - 1$ without the use of any failure detector, and therefore reach a contradiction [8,9,18].

It follows from the definition of ζ_F that there exists a subset of $(2n-1)$ processes, G for which there is no subset $H \subset G$ of n processes such that $\zeta(H) \in \zeta_F$. In order to solve SB for $m = 2n - 1$, map the process names in Π onto the names in G. Then, using the processes' mapped names, execute algorithm $\mathcal{A}$. Whenever a failure detector $\zeta(S) \in \zeta_F$ is queried in $\mathcal{A}$, choose some process $p \in S - G$ as its output. There exists such a process p, since otherwise, $S \subset G$, $|S| = n$ and $\zeta(S) \in \zeta_F$, in contradiction to the definition of G. The name p is a valid output for $\zeta(S)$ in our simulation, since it is not in the participating set in the simulation, and therefore not a correct process in the simulation. Thus, $\mathcal{A}$ could be wait-free simulated using only read/write registers, and the outputs of the processes in $\mathcal{A}$ can be taken as their outputs for the SB task. If n reach a decision by simulating steps of $\mathcal{A}$, at least one of them outputs 0 and at least one other outputs 1, as required. □

Unfortunately, finding the size of the minimal set R of subsets of size r, such that each subset of size k contains at least one of the sets in R is in itself an open problem in combinatorics, closely related to the covering design task [19,25,28]. Therefore, calculating an exact numeric value for the lower bound induced by Theorem 22 is an open problem[3] . However, for some m and n, one can find a smaller set of ζ's than the size used by Algorithm 2 that is still eligible by the condition of Theorem 22. For example, when $m = 9$ and $n = 4$, ζ-Set consists of 6 ζ's, while only 5 are required by the lower bound (due to [19]). This disparity between the lower and upper bound may imply that a better algorithm than Algorithm 2 in terms of the number of ζ's used is yet to be found.

9 $\zeta(s)$ and the k-anti-Ω Hierarchy

In this section, we briefly discuss the power of a single $\zeta(S)$ failure detector in comparison with other well known failure detectors. We first consider the environment $\mathcal{E}^m$ that contains all possible failure patterns. Then, in Section 9.2, we compare the relative power of a single $\zeta(S)$ in $\mathcal{E}^n$ where $n < m$.

Trivially, if $|S| = m$, then there is only one possible $\zeta(S)$, and it has the same behavior as anti-Ω, and therefore, can solve the $(m-1)$-set consensus task.

If $|S| < m$, we show that $\zeta(S)$ implements anti-Ω, and therefore allows for solving $(m-1)$-set consensus in $\mathcal{E}^m$, but not $(m-2)$-set consensus. However, we show that Ω, which is a failure detector powerful enough to solve the $(m-2)$-set consensus task and, in fact, even the consensus task, cannot implement any $\zeta(S)$ for $1 < |S| < m$ in $\mathcal{E}^m$.

9.1 $\zeta(S)$ in $\mathcal{E}^m$

Theorem 23 *In $\mathcal{E}^m$, $\zeta(S)$, where $S \subset \Pi$ and $1 < |S| \leq m$, implements anti-Ω.*

Proof. One can implement anti-Ω by taking the output of $\zeta(S)$ verbatim. By definition, $\zeta(S)$ eventually returns only the names of processes in S. If there are no correct processes in S, any such output by $\zeta(S)$ is a valid output of anti-Ω. If however, there is a correct process in S, at least one of the correct processes in S, p is returned only a finite number of times by $\zeta(S)$. Therefore, by using the output of $\zeta(S)$ as the output of anti-Ω, there is at least one correct process (p) that is returned only a finite number of times, as required by the definition of anti-Ω. □

Since any $\zeta(S)$ is able to solve $(m-1)$-set consensus in $\mathcal{E}^m$ for all $1 < |S| \leq m$, it is natural to inquire whether it can do better and solve the $(m-2)$-set consensus task. We show first that when $|S| \leq m$, $\zeta(S)$ is unable to solve the stronger task.

Theorem 24 *In $\mathcal{E}^m$, no failure detector $\zeta(S)$, where $S \subseteq \Pi$ and $1 < |S| \leq m$, can solve the $(m-2)$-set consensus task.*

Proof. Let $\Pi = \{p_1, \ldots, p_m\}$, as defined in Section 3.

Consider first the case $|S| = m$, i.e., $S = \Pi$. Recall that anti-Ω cannot solve $(m-2)$-set consensus [15]. Thus, by Theorem 23, $\zeta(\Pi)$ cannot solve $(m-2)$-set consensus either.

Now consider any subset $S \subset \Pi$, $|S| = n < m$. Assume, by contradiction, that there exists an algorithm $\mathcal{A}$ that uses $\zeta(S)$ to solve the $(m-2)$-set consensus task. Without loss of generality, assume that $S = \{p_1, \ldots, p_n\}$ (if S is another n subset of Π, processes in Π could be renamed so that this assumption holds).

[3]However its value is bounded between $\nu = \binom{m}{2n-1}/\binom{m-n}{(2n-1)-n} = \binom{m}{n}/\binom{2n-1}{n}$, which is the number of sets of size $2n-1$ divided by the number of sets of this size covered by each ζ, and $\nu \cdot [1 + ln(\binom{m-n}{m-(2n-1)})]$ due to [25].

We show that following our previous assumption, it is possible to solve the $(m-2)$-set consensus task in read/write among $m-1$ processes, $q_1, \ldots, q_{m-1}$, leaving us with a contradiction. Map the processes $q_1, \ldots, q_{m-1}$ to $p_2, \ldots, p_m$, by mapping each q_i to p_{i+1}. Now, run algorithm $\mathcal{A}$ using only read/write registers, and whenever algorithm $\mathcal{A}$ queries $\zeta(S)$, simulate the output of $\zeta(S)$ by simply returning p_1. Since no process q_i is mapped to p_1, p_1 is not participating in the simulation, and therefore it is a valid output for $\zeta(S)$.

The result of running algorithm $\mathcal{A}$ is having the processes decide on $m-2$ names taken from $p_2, \ldots, p_m$, which can be simply mapped back to $q_1, \ldots, q_{m-1}$. Thus, we have read/write implemented $(m-2)$-set consensus for $(m-1)$ processes, and achieved the required contradiction. □

The last result may suggest that in tightly named systems, a given $\zeta(S)$ is weaker than any failure detector, $\mathcal{D}$, capable of solving the $(m-2)$-set consensus task. However, this is not the case. Indeed, consider the $\neg\Omega_k$ set of failure detectors. As mentioned earlier, it was shown that $\neg\Omega_{m-1}$ is equivalent to anti-Ω in every environment and thus can implement $m-1$ set consensus, and that $\neg\Omega_1$ is equivalent to Ω and thus can solve consensus.

Moreover, any failure detector $\neg\Omega_k$ can implement $\neg\Omega_r$, when $r \geq k$. Of particular interest to us is $\neg\Omega_{m-2}$, which is able to solve the $(m-2)$-set consensus task in $\mathcal{E}^m$, but as we show next, cannot implement $\zeta(S)$. Moreover, in Theorem 25 we prove a stronger result: in tightly named systems, Ω itself (that is $\neg\Omega_1$) cannot implement $\zeta(S)$, if $|S| < m$.

Theorem 25 *In $\mathcal{E}^m$, Ω cannot implement $\zeta(S)$ for any $1 < |S| < m$, $S \subset \Pi$.*

Proof. Without loss of generality, assume that $\Pi = \{p_1, \ldots, p_m\}$ and $S = \{p_1, \ldots, p_n\}$ where $n = |S| < m$ (otherwise, we could always rename the processes in Π). We turn to the game theory approach presented by Zieliński [29], in order to prove that Ω cannot implement $\zeta(S)$. In the framework of [29], two players NO and YES play a game, where NO takes the first turn. In turn i, NO outputs (C_i, Z_i), such that C_i is a set of process names, $C_i \subset C_{i-1}$, $Z_i \subseteq Z_{i-1}$ (when $i > 1$), and $Z_i \in \mathit{infset}_\Omega(C_i)$. Following NO, YES outputs a set, T_i, such that $T_i \subseteq T_{i-1}$ (when $i > 1$) and $T_i \in \mathit{infset}_{\zeta(S)}(C_i)$. The first player unable to take a move loses the game. YES has a winning strategy if, and only if, Ω can implement $\zeta(S)$. We provide a winning strategy for NO, and thus prove that the latter failure detector cannot be implemented by the former.

The strategy for NO is rather simple. Since $|S| < m$, we are assured that $p_m \notin S$. Let $C_1 = \Pi$, and let $Z_1 = \{p_m\}$. Since $p_m \in C_1$, it is a correct process that Ω's output can eventually stabilize on, and therefore, $Z_1 \in \mathit{infset}_\Omega(C_1)$. Throughout the game, NO keeps $p_m \in C_i$ and $Z_i = Z_1$ for each i, and thus, $Z_i \in \mathit{infset}_\Omega(C_i)$. Our strategy enables NO to choose such C_i and Z_i, while causing YES to decrease the size of T_i at each move. When YES is left with only one process in T_i, it cannot take any further moves, and loses.

At YES's first move, T_1 cannot equal S, since $S \subset C_1$, and therefore there must be at least one process $q^1 \in S$, such that $q^1 \notin T_1$ in order for $T_1 \in \mathit{infset}_{\zeta(S)}(C_1)$. Note that there is no reason for YES to eliminate more than one process from S in order to generate T_1, as it can always do so in its next move. Therefore we may assume that $T_1 = S \setminus q^1$.

As a response, NO can output $C_2 = C_1 \setminus q^1$, $Z_2 = Z_1$. Since $p_m \notin S$ while $q^1 \in S$, we have $p_m \neq q^1$. Therefore $p_m \in C_2$, and our prior requirements still hold. YES now has to respond with $T_2 \subset T_1$, since it can no longer play T_1 as $C_2 \cap S = (C_1 \setminus q^1) \cap S = S \setminus q^1 = T_1$. Therefore the best response by YES is to choose $q^2 \in T_1$ and have $T_2 = T_1 \setminus q^2$. By induction, after each move by YES, $T_i = T_{i-1} \setminus q^i$, NO can respond with $C_{i+1} = C_i \setminus q^i$, $Z_{i+1} = \{p_m\}$. Since $|C_1| = m > n > n-1 = |T_1|$, and in each turn, NO

eliminates from C_i a process previously eliminated by YES from T_i, YES is the first to run out of moves, and therefore, this is a winning strategy for NO. $\square$

9.2 $\zeta(S)$ in $\mathcal{E}^n$

Since Ω cannot implement $\zeta(S)$ for any S (Theorem 25), no $\neg\Omega_k$, $1 \leq k \leq m-1$, can implement $\zeta(S)$. This observation, combined with Theorem 24, looks somewhat paradoxical. On the one hand, we have shown that for any S, $\zeta(S)$ cannot implement $(m-2)$-set consensus in $\mathcal{E}^m$. On the other hand, we proved that Ω, a failure detector capable of solving the consensus task, is unable to implement $\zeta(S)$, in such systems. As a result, in $\mathcal{E}^m$, there does not exist a hierarchy that totally orders "eventual" failure detectors. However, in $\mathcal{E}^n$ where $n < m$, and given a failure detector $\zeta(S)$, either a specific $\neg\Omega_k$ failure detector is capable of implementing $\zeta(S)$, or it can be wait-free implemented using only read/write registers, as proved below.

Theorem 26 *Consider the environment $\mathcal{E}^n$, where $n < m$. Let $S \subset \Pi$ be a subset of n processes.*
(a) for $1 \leq r \leq n-1$, $\neg\Omega_r$ implements $\zeta(S)$.
(b) for $n \leq r \leq m-1$, $\neg\Omega_r$ can be wait-free implemented using only read/write registers.

Proof. (a) It suffices to show that $\neg\Omega_{n-1}$ implements $\zeta(S)$. When $r < (n-1)$, $\neg\Omega_r$ implements $\neg\Omega_{n-1}$, and therefore the other cases trivially follow. Whenever queried, $\neg\Omega_{n-1}$ returns a set of $m-n+1$ process names. As $|\Pi \setminus S| = m-n$, at least one of the processes returned by $\neg\Omega_{n-1}$, p, is in S.

In order to implement $\zeta(S)$, the processes keep track of the current participating set, P. If $P \neq S$, there is a process $q \in S$ such that $q \notin P$, that can be returned as the output of $\zeta(S)$. If, on the other hand, $P = S$, we take any $p \in S$, returned by $\neg\Omega_{n-1}$, as the output of S. Such p exists as explained above.

We now show that the above implementation of $\zeta(S)$ is valid. Once no new processes wake up, P stabilizes. If $P \neq S$, we infinitely return a process which is not in the participating set, and therefore not correct, as (a valid) output of $\zeta(S)$. If, however, $P = S$, since $|S| = n$, no process in $\Pi \setminus S$ may participate, according to our assumption on the maximum participating set size. Therefore, no process in $\Pi \setminus S$ can be correct, and if p is the only correct process in S, it is the only correct process in Π. Thus, if $p \in S$ is returned an infinite number of times by $\neg\Omega_{n-1}$, it cannot be the only correct process in S, or otherwise the output of $\neg\Omega_{n-1}$ would be invalid. Therefore we may return a process p as defined above, as a valid output of $\zeta(S)$.

(b) It suffices to show that $\neg\Omega_n$ can be implemented in read/write. When $r > n$, $\neg\Omega_n$ implements $\neg\Omega_r$ and the other cases trivially follow. Whenever queried, $\neg\Omega_n$ returns $m-n$ processes. As there are at most n participating processes, we can keep track of the participating set, and return the names of (the first) $m-n$ processes that have not yet taken a step as the output of $\neg\Omega_n$. Since we are not returning names of correct processes, this is a valid output for $\neg\Omega_n$. $\square$

Figure 1 summarizes the results presented in this section. In $\mathcal{E}^m$, $\zeta(S)$ is not part of the hierarchy formed by the $\neg\Omega_k$ failure detectors, while in $\mathcal{E}^n$, $n < m$, it is weaker than any non read/write implementable failure detector in that hierarchy.

Another interesting result is that in $\mathcal{E}^n$, anti-Ω is a trivial failure detector while $\zeta(S)$ is non-trivial. On the other hand, in $\mathcal{E}^m$, since Ω implements anti-Ω but does not implement $\zeta(S)$, anti-Ω cannot implement $\zeta(S)$. Therefore, in both cases, anti-Ω is strictly weaker than $\zeta(S)$ (for any S).

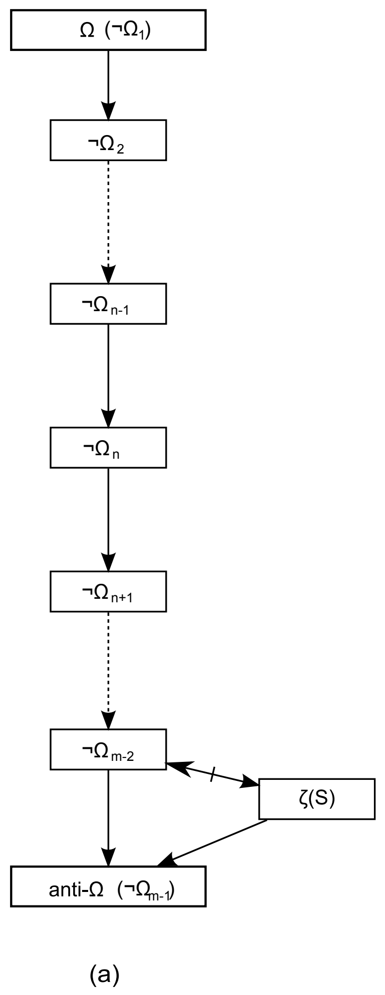

Figure 1: Implementation of $\zeta(S)$ and $\neg\Omega_k$ failure detectors in $\mathcal{E}^m$, and in $\mathcal{E}^n$, when $n < m$. An arrow from failure detector $\mathcal{A}$ to failure detector B signifies that B can be implemented using $\mathcal{A}$. (a) In $\mathcal{E}^m$, any $\zeta(S)$ implements anti-Ω but does not implement $\neg\Omega_{m-2}$, and cannot be implemented by Ω. (b) In $\mathcal{E}^n$ $(n < m)$, $\zeta(S)$ can be implemented by $\neg\Omega_{n-1}$ (grayed failure detectors can be implemented in $\mathcal{E}^n$ using only reads and writes).

10 Concluding Remarks

In this paper, we focused on a specific environment $\mathcal{E}^n$ that describes runs in which at most n out of m processes participate. In this environment we determined a weakest family of failure detectors for the task of renaming in the class of **Z1–Z3**: each failure detector in **Z1–Z3** that solves renaming or symmetry breaking can be used to implement some failure detector in ζ_n.

In particular, we presented an algorithm that uses a subset of the detectors in ζ_n to solve the symmetry breaking task. On the other hand, we show that *all* detectors in ζ_n are required to solve the $(n-1)$-set consensus task. This serves as an interesting distinction between the case $n = m$ (all processes may participate) and $n < m$ (not all processes participate), where in the former the weakest non-trivial failure detector in **Z1–Z3** can be used to solve the $(n-1)$-set consensus task. Intriguingly, there is no single weakest detector in ζ_n and no detector in this family can be implemented by the combined power of the others.

It may look surprising that renaming does not have a weakest failure detector in **Z1** – **Z3**. Indeed, Jayanti and Toueg has recently shown that any problem (within a very general definition of what a problem is that covers renaming and symmetry breaking) has a weakest failure detector [24]. But there is no contradiction here, applying the method by which weakest failure detector is constructed in [24] to the members of ζ_n we obtain a failure detector that is not in **Z1** – **Z3**, in particular it is not eventual (does not satisfy **Z1**). In fact, as we have shown in this paper, the weakest failure detectors for renaming and symmetry breaking do exist and they are renaming and symmetry breaking themselves: both these problems can be viewed as failure detectors that satisfy **Z2** and **Z3** (but of course not **Z1**).

Therefore, **Z1** is necessary in order to obtain our result. But are the other two assumptions, finite ranges of the failure detector output (**Z2**) and independence of the timing of failures (**Z3**), necessary for the derivation of our result? **Z2** looks quite natural and is in fact a standard assumption that is used in many recent simulation-based derivation of failure detectors [15, 30]. But can we get rid of **Z3**? For example, $\neg\Omega$ is shown in [30] to be the weakest failure detector for $(m-1)$-set consensus assuming only **Z2**. Can we reuse the arguments of [30] to show that ζ_n is the weakest failure-detector family for renaming in **Z1** – **Z3**? This is a very appealing open question that does not seem to have a straightforward answer (at least we did not manage to resolve after several attempts).

Another interesting unresolved question that merits further investigation concerns the discrepancy between the lower and upper bound of the number of ζ_n's failure detectors necessary to implement the symmetry breaking task in $\mathcal{E}^n$. Is there a more efficient algorithm than Algorithm 2? Failing to find a better algorithm, is there a way to increase the lower bound?

Moreover, having a lower bound which is greater than a single failure detector out of ζ_n in $\mathcal{E}^n$, alludes to the possibility of an interesting task still weaker than symmetry breaking. Could such a task be found and declared using the mechanisms provided in this paper?

Finally, what is the power of the failure detectors in ζ_n in $\mathcal{E}^m$? We have shown that in $\mathcal{E}^m$, any single failure detector of $\zeta(S) \in \zeta_n$ can implement $(m-1)$-set consensus, but fails to implement the $(m-2)$-set consensus task. Yet, $\zeta(S)$ is strictly stronger than anti-Ω. Is there any task which can be solved using $\zeta(S)$ which cannot be solved using anti-Ω in $\mathcal{E}^m$?

Acknowledgments

We are grateful to Eli Gafni, Piotr Zieliński and the anonymous referees of DISC and this journal, for many fertile discussions, helping us to strengthen the results and improve the presentation.

References

[1] Yehuda Afek and Israel Nir. Failure detectors in loosely named systems. In *PODC*, pages 65–74, 2008.

[2] Hagit Attiya, Amotz Bar-Noy, Danny Dolev, David Peleg, and Rüdiger Reischuk. Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524–548, 1990.

[3] Hagit Attiya and Arie Fouren. Polynomial and adaptive long-lived (2k-1)-renaming. In *DISC*, pages 149–163, 2000.

[4] Hagit Attiya and Sergio Rajsbaum. The combinatorial structure of wait-free solvable tasks. *SIAM Journal on Computing*, 31(4):1286–1313, 2002.

[5] Hagit Attiya and Jennifer Welch. *Distributed Computing. Fundamentals, Simulations, and Advanced Topics.* McGraw-Hill, 1998.

[6] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC*, pages 91–100. ACM Press, 1993.

[7] Armando Castañeda and Sergio Rajsbaum. New combinatorial topology upper and lower bounds for renaming. In *PODC*, pages 295–304, 2008.

[8] Armando Castañeda and Sergio Rajsbaum. New combinatorial topology upper and lower bounds for renaming: The lower bound. *Distributed Computing*, 52(5-6):287–301, 2010.

[9] Armando Castañeda and Sergio Rajsbaum. New combinatorial topology upper and lower bounds for renaming: The upper bound. *Journal of the ACM (to appear)*, 2012.

[10] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[11] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[12] Soma Chaudhuri. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In *PODC*, pages 311–324, 1990.

[13] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[14] Eli Gafni. Renaming with k-set-consensus: An optimal algorithm into $n + k$ - 1 slots. In *OPODIS*, pages 36–44, 2006.

[15] Eli Gafni and Petr Kuznetsov. On set consensus numbers. *Distributed Computing*, 24(3-4):149–163, 2011.

[16] Eli Gafni, Achour Mostéfaoui, Michel Raynal, and Corentin Travers. From adaptive renaming to set agreement. *Theoretical Computer Science*, 410:1328–1335, 2009.

[17] Eli Gafni and Sergio Rajsbaum. Distributed programming with tasks. In *OPODIS*, pages 205–218, 2010.

[18] Eli Gafni, Sergio Rajsbaum, and Maurice Herlihy. Subconsensus tasks: Renaming is weaker than set agreement. In *International Symposium on Distributed Computing*, pages 329–338, 2006.

[19] Daniel M. Gordon, Greg Kuperberg, and Oren Patashnik. New constructions for covering designs. *J. Combin. Designs*, 3:269–284, 1995.

[20] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):123–149, 1991.

[21] Maurice Herlihy and Sergio Rajsbaum. Algebraic spans. *Mathematical Structures in Computer Science*, 10(4):549–573, 2000.

[22] Maurice Herlihy and Nir Shavit. The asynchronous computability theorem for t-resilient tasks. In *STOC*, pages 111–120, 1993.

[23] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(2):858–923, 1999.

[24] Prasad Jayanti and Sam Toueg. Every problem has a weakest failure detector. In *PODC*, pages 75–84, 2008.

[25] László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[26] Michel Raynal. K-anti-Omega, August 2007. Rump session at PODC 2007.

[27] Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. In *STOC*, pages 101–110. ACM Press, 1993.

[28] Johanan Schönheim. On coverings. *Pacific Journal of Mathematics*, pages 1405–1411, 1964.

[29] Piotr Zielinski. Automatic classification of eventual failure detectors. In *DISC*, pages 465–479, 2007.

[30] Piotr Zieliński. Anti-*omega*: the weakest failure detector for set agreement. *Distributed Computing*, 22(5-6):335–348, 2010.