On Failure Detectors Weaker Than Ever

Wei Chen
Microsoft Research Asia
weic@microsoft.com

Yu Chen
Microsoft Research Asia
ychen@microsoft.com

Jialin Zhang
Tsinghua University
zhanggl02@mails.tsinghua.edu.cn

Abstract

In this paper, we apply the partition approach proposed in [6] to study weak failure detectors for set agreement problem for the shared-memory model. In a system with $n+1$ processes, for any $2 \leq k \leq n$, we first propose a partitioned failure detector $\Pi\Omega_k$ that solves k-set agreement with shared read/write registers and is strictly weaker than Ω_k, which was conjectured to be the weakest failure detector for k-set agreement in the shared-memory model [16]. We then propose a series of partitioned failure detectors that can solve n-set agreement, yet they are strictly weaker than Υ [8], the weakest failure detector ever found before our work to circumvent any asynchronous impossible problems in the shared-memory model. Our results not only lower the upper bound on the failure detectors for set agreement, but also further demonstrate the power of the partition approach. They strongly reinforce the statement we made in [6] that the partition approach opens a new dimension for weakening failure detectors related to set agreement, and it is an effective test to check whether a failure detector is the weakest one or not for set agreement. So far, no candidates for the weakest failure detectors of set agreement pass our partition test.

MSR-TR-2007-50

1 Introduction

Failure detector abstractions are first proposed by Chandra and Toueg in [3] to circumvent the impossibility result of consensus [7], and have since become a powerful technique to encapsulate system conditions needed to solve many distributed computing problems. Among them the problem of k-set agreement has received many attention from the research community. Informally, in k-set agreement each process proposes some value and eventually all correct processes (those that do not crash) decide on at most k different values [4]. It has been shown that k-set agreement cannot be solved in asynchronous systems when k or more processes may crash [1, 10, 17]. In recent years, a number of studies have focused on failure detectors for solving k-set agreement problem [18, 15, 9, 13, 14, 8, 6]. These studies form the collective effort in the pursuit of the weakest failure detector for k-set agreement, a goal yet to be reached. A particular candidate Ω_k was conjectured to be the weakest failure detector for k-set agreement [16].

Consider distributed shared-memory model with $n+1$ processes. In a very recent paper [8], Guerraoui et.al define a new class of failure detectors Υ and show that among a wide range of failure detectors defined as *eventually stable failure detectors*, Υ is the weakest one needed to solve *any* impossible problem in shared-memory distributed systems, and Υ solves the n-set agreement problem. The Υ failure detector disproves the conjecture on Ω_k for the case of $k = n$. For a general k, a generalized Υ^k is proposed to solve k-set agreement, but only when at most k processes may crash, so it does not disprove the conjecture on Ω_k for wait-free k-set agreement.

The eventually stable failure detectors encompass most failure detectors known to solve distributed decision tasks in the shared-memory model prior to [8], as the authors claimed. Therefore, as the title of their paper says, indeed Υ is the weakest failure detector ever found that solves any impossible problem in distributed computing.

However, parallel to the work of [8], we introduce in [6] the partition approach to failure detectors, and propose two new classes of failure detectors Π_k and Π_k^S ($1 \leq k \leq n$) that we call partitioned failure detectors. We show that they are strong enough to solve k-set agreement in the message-passing model, but are strictly weaker than other failure detectors for k-set agreement known at the time (not including Υ).

In the partition approach, failure detectors partition the processes into multiple components and only processes in one of the component (called a *live component*) are required to satisfy all safety and liveness properties, while processes in other components only need to satisfy safety properties. Since those processes in non-live components may generate quite arbitrary failure detector outputs, intuitively the partitioned failure detectors are a new breed that does not fall into the eventually stable failure detectors covered by [8]. In this paper we verify this intuition and show that by using the partition approach we are able to find a series of new failure detectors even weaker than Υ but are still strong enough to solve n-set agreement. We also apply the partition approach to show that Ω_k is not the weakest failure detector for wait-free k-set agreement for any $k \geq 2$.

Our results are developed in several stages. First, by a simple modification to Π_k, we obtain a new class of failure detector $\Pi\Omega_k$. We show that $\Pi\Omega_k$ is strong enough to solve k-set agreement with shared read/write registers but it is not comparable with Υ, for all $k = 2, 3, \ldots, n$. One direct consequence is that $\Pi\Omega_k$ is strictly weaker than Ω_k (because Ω_k is stronger than Υ), which disproves the conjecture that Ω_k is the weakest failure detector for wait-free k-set agreement in the shared-memory model for any $k \geq 2$. Moreover, $\Pi\Omega_k$ is the first failure detector class that solves k-set agreement (for generic k) but is incomparable with Υ (a result in [6] implies that Π_k and Π_k^S are stronger than Ω_n, which is strictly stronger than Υ). For example, even though failure detector $\Pi\Omega_2$ solves 2-set agreement, it is not stronger than Υ.

Second, we mix some of the properties of $\Pi\Omega_k$ and Υ and define another class of partitioned failure

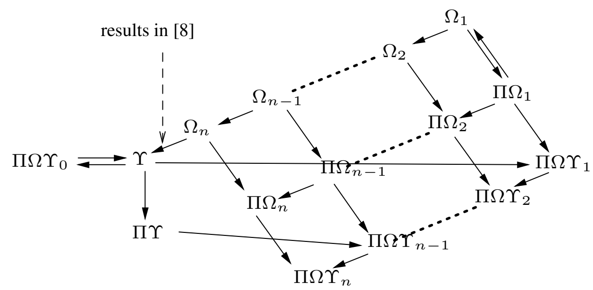

Figure 1: Relationship diagram for failure detectors ($n \geq 3$). If $A \rightarrow B$, then A can be transformed into B. If there is no directed path from A to B, then A cannot be transformed into B (Footnote 1 contains the only exceptions).

detectors $\Pi\Omega\Upsilon_k$, and we show that for any $k \geq 1$, $\Pi\Omega\Upsilon_k$ can still solve n-set agreement but it is strictly weaker than both $\Pi\Omega_k$ and Υ. Moreover, as k increases, the strength of $\Pi\Omega\Upsilon_k$ is strictly weakened. Hence, we find a family of n different failure detector classes strictly weaker than Υ, which is the weakest one ever found before our work.

Finally, to demonstrate the power of partition approach, we apply the approach directly to Υ, and define a partitioned version of it denoted as $\Pi\Upsilon$. We again show that $\Pi\Upsilon$ is strong enough to solve n-set agreement but is strictly weaker than Υ. Moreover, $\Pi\Upsilon$ is incomparable with $\Pi\Omega\Upsilon_k$ for any $k \leq n-3$ (or $k = 2$ and n is odd) but is strictly stronger than $\Pi\Omega\Upsilon_{n-1}$. This demonstrates that there are more than one way to weaken Υ using the partition approach.

Figure 1 characterizes the exact relationship among all failure detectors we proposed in this paper and the previously defined ones Ω_k and Υ. Note that every nonexistent directed path in the figure corresponds to an impossible transformation from the source class to the destination class, with only a couple of exceptions.[1] Since Υ is already very weak, one can imagine that it would be very delicate to define and prove that so many failure detector classes are incomparable to or strictly weaker than Υ. Indeed, the definitions of failure detectors are subtle, and the proofs of the impossible transformations are the most delicate and technically involved.

Our results not only show a number of new failure detectors that are strictly weaker than Υ, but more importantly, they further demonstrate the power of the partition approach. They strongly reinforce the statement we made in [6] that the partition approach opens a new dimension for weakening various failure detectors related to set agreement, and it is an effective test to check whether a failure detector could be the weakest one solving set agreement or not. Using the approach, we have successfully shown that (1) $\Omega_k \times \Sigma$ is not the weakest failure detector for k-set agreement in the message-passing model for any $k \geq 2$ (in [6]), (2) Ω_k is not the weakest failure detector for k-set agreement in the shared-memory model for any $k \geq 2$, and (3) Υ is not the weakest failure detector for n-set agreement in the shared-memory model. So far, no failure detectors that were considered as the candidates for the weakest failure detectors for set agreement have passed our partition test. Therefore, we believe that it is important to use the partition approach as an effective research tool in our pursuit to the ultimate weakest failure detectors for set agreement.

The rest of the paper is organized as follows. Section 2 provides the model. Section 3 defines $\Pi\Omega_k$ and show how it solves k-set agreement. Section 4 and 5 define $\Pi\Omega\Upsilon_k$ and $\Pi\Upsilon$ and shows how they solves n-set agreement, respectively. Section 6 provides a central place to show the relationship among all failure detectors as captured by Figure 1. We conclude the paper in Section 7. Most algorithms and their proofs are included in the main text, while the appendix includes several more technically-involved proofs.

[1] The exceptions are the following two problems that are still open: (a) whether $\Pi\Omega_k$ can be transformed into $\Pi\Omega\Upsilon_{k-1}$ for any $k \geq 2$; and (b) whether $\Pi\Upsilon$ can be transformed into $\Pi\Omega\Upsilon_{n-2}$ when n is even.

2 Model

We consider asynchronous shared memory distributed systems augmented with failure detectors. Our model is the same as the model in [8], which is based on the models of [11, 12, 2]. We provide the necessary details of the model below.

We consider a system with $n+1$ processes $P = \{p_1, p_2, \ldots, p_{n+1}\}$ where $n \geq 1$. Let $\mathcal{T}$ be the set of global time values, which are non-negative integers. Processes do not have access to the global time. A *failure pattern* F is a function from $\mathcal{T}$ to 2^P, such that $F(t)$ is the set of processes that have failed by time t. Failed processes do not recover, i.e., $F(t) \subseteq F(t+1)$ for all $t \in \mathcal{T}$. Let *correct*(F) denote the set of *correct processes*, those that do not crash in F. A process is *faulty* if it is not correct. A *failure detector history* H is a function from $P \times \mathcal{T}$ to an output range $\mathcal{R}$, such that $H(p,t)$ is the output of the failure detector module of process $p \in P$ at time $t \in \mathcal{T}$. A *failure detector* $\mathcal{D}$ is a function from each failure pattern to a set of failure detector histories, representing the possible failure detector outputs under failure pattern F.

Processes communicate with each other by writing to and reading from shared atomic registers. A deterministic algorithm A using a failure detector $\mathcal{D}$ is a collection of $n+1$ deterministic automata, one for each process. Processes executes by taking *steps*. In each step, a process p: (a) reads from a shared register to obtain a value, or writes a value to a shared register, or queries its failure detector module, based on its current local state; and (b) transitions its current state to a new state, based on its current state, the value returned from the read or from the failure detector module, and the algorithm automaton on p. Each step is completed at one time point t, but the process may crash in the middle of taking its step. A *run* of algorithm A with failure detector $\mathcal{D}$ under a failure pattern F is an infinite sequence of steps such that every correct process takes an infinite number of steps and no faulty process takes any step after it crashes.

We say that a failure detector class $\mathcal{C}_1$ is *weaker than* a failure detector class $\mathcal{C}_2$, if there is a transformation algorithm T such that using any failure detector $\mathcal{D}_2 \in \mathcal{C}_2$, algorithm T implements a failure detector $\mathcal{D}_1 \in \mathcal{C}_1$. By implementing $\mathcal{D}_2$ we mean that for any run of algorithm T with failure detector $\mathcal{D}_2$ under a failure pattern F, T generates the outputs of $\mathcal{D}_1$ as a distributed variable $\mathcal{D}_1$-*output* such that there exists failure detector history $H \in \mathcal{D}_1(F)$ and $H(p,t) = \mathcal{D}_1$-*output*(p,t) for all $p \in P$ and all $t \in \mathcal{T}$, where $\mathcal{D}_1$-*output*(p,t) is the value of the variable $\mathcal{D}_1$-*output* on p at time t. If $\mathcal{C}_1$ is weaker than $\mathcal{C}_2$, we denote it as $\mathcal{C}_1 \preceq \mathcal{C}_2$ and also refer to it as $\mathcal{C}_2$ can be transformed into $\mathcal{C}_1$. if $\mathcal{C}_1 \preceq \mathcal{C}_2$ and $\mathcal{C}_2 \not\preceq \mathcal{C}_1$, we say that $\mathcal{C}_1$ is *strictly weaker than* $\mathcal{C}_2$ and denote it as $\mathcal{C}_1 \prec \mathcal{C}_2$. If $\mathcal{C}_1 \preceq \mathcal{C}_2$ and $\mathcal{C}_2 \preceq \mathcal{C}_1$, we say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are equivalent and denote it as $\mathcal{C}_1 \equiv \mathcal{C}_2$.

In k-set agreement with $1 \leq k \leq n$, each process proposes a value, and makes an irrevocable decision on one value. It needs to satisfy the following three properties: (1) *Validity*: If a process decides v, then v has been proposed by some process. (2) *Uniform k-Agreement*: There are at most k different decision values. (3) *Termination*: Eventually all correct processes decide.

Two related failure detector classes are Ω_k and Υ. Failure detectors in Ω_k output a subset of P of size at most k, and there is a time after which all processes always output the same nonempty set, which contains at least one correct processes. Failure detectors in Υ also output a subset of P, and there is a time after which all processes always output the same nonempty set, which is not exactly the set of correct processes.

3 Failure Detector $\Pi\Omega_k$

Our first step is to modify the statically partitioned failure detector class Π_k defined in [6] for the message-passing model so that it solves k-set agreement in the shared-memory model but is incomparable with Υ. The modification is a simple one by replacing the quorum output of Π_k with a component ID *cid*.

3.1 Specification of $\Pi\Omega_k$

The output of $\Pi\Omega_k$ for process p is a tuple $(\mathit{isLeader}, \mathit{lbound}, \mathit{cid})$, where *isLeader* is a boolean value indicating whether this process is a leader or not, *lbound* is a non-negative integer indicating the upper bound on the number of possible leaders in p's partitioned component, and *cid* is a component ID drawn from an ID set $\mathcal{I}$ or is a special value $\perp \notin \mathcal{I}$. The *cid* output indicates the component the process belongs to and could be $\perp$ for an initial period before the failure detector decides on a partition.[2]

For a failure detector output x, we use $x.v$ to denote the field v of x, where v could be *isLeader*, *lbound*, or *cid* in the case of $\Pi\Omega_k$. We say that a process p is an *eventual leader* (under a failure pattern F and a failure detector history H) if p is correct and there is a time after which the *isLeader* output on p is always *True*.

A *partition* of P is $\pi = \{P_1, \ldots, P_s\}$, where $s \geq 1$ and P_i's are non-empty subsets of P such that they do not intersect with one another and their union is P. For a process p, we use $\pi[p]$ to denote the partitioned component that contains p. For a component $P_j \subseteq P$ (under a failure pattern F and a failure detector history H), we define $\mathit{lbound}(P_j) = \max\{H(p,t).\mathit{lbound} \mid t \in \mathcal{T}, p \in P_j \setminus F(t)\}$,[3] and $\mathit{Leaders}(P_j) = \{p \in P_j \cap correct(F) \mid \exists t, \forall t' > t, H(p,t').\mathit{isLeader} = \mathit{True}\}$. The value $\mathit{lbound}(P_j)$ is the maximum *lbound* value among processes in component P_j, while $\mathit{Leaders}(P_j)$ is the set of eventual leaders in P_j.

A failure detector $\mathcal{D}$ is in the class $\Pi\Omega_k$ if for any failure pattern F and any failure detector history $H \in \mathcal{D}(F)$, there exists a partition $\pi = \{P_1, \ldots, P_s\}$ of P, such that the following properties hold. First, the *cid* output needs to satisfy these properties:

($\Pi C1$) The *cid* outputs on all correct processes eventually always output non-$\perp$ values. Formally, $\exists t_0 \in \mathcal{T}, \forall p \in \mathit{correct}(F), \forall t \geq t_0, H(p,t).\mathit{cid} \neq \perp$.

($\Pi C2$) The non-$\perp$ *cid* outputs distinguish different components. Formally, $\forall t_1, t_2 \in \mathcal{T}, \forall p_1 \notin F(t_1), \forall p_2 \notin F(t_2), (H(p_1,t_1).\mathit{cid} \neq \perp \wedge H(p_2,t_2).\mathit{cid} \neq \perp) \Rightarrow ((H(p_1,t_1).\mathit{cid} = H(p_2,t_2).\mathit{cid}) \Leftrightarrow (\pi[p_1] = \pi[p_2]))$.

Next, the *isLeader* and *lbound* outputs satisfy the following set of safety and liveness properties. The safety property is:

($\Pi\Omega1$) The sum of the maximum *lbound* outputs in all partitioned components does not exceed k. Formally, $\sum_{j=1}^{s} \mathit{lbound}(P_j) \leq k$.

The liveness part specifies that there exists one partitioned component P_j such that:

($\Pi\Omega2$) Eventually *lbound* outputs by all processes in P_j are the same. Formally, $\exists t_0 \in \mathcal{T}, \forall t_1, t_2 \geq t_0, \forall p_1 \in P_j \setminus F(t_1), \forall p_2 \in P_j \setminus F(t_2), H(p_1,t_1).\mathit{lbound} = H(p_2,t_2).\mathit{lbound}$.

($\Pi\Omega3$) Eventually the *isLeader* outputs on any correct process in P_j do not change. Formally, $\exists t_0 \in \mathcal{T}, \forall t > t_0, \forall p \in P_j \setminus F(t), H(p,t).\mathit{isLeader} = H(p,t_0).\mathit{isLeader}$.

($\Pi\Omega4$) There is at least one eventual leader. Formally, $|\mathit{Leaders}(P_j)| \geq 1$.

($\Pi\Omega5$) The number of eventual leaders is eventually bounded by the *lbound* outputs. Formally, $\exists t_0 \in \mathcal{T}, \forall t \geq t_0, |\mathit{Leaders}(P_j)| \leq H(p,t).\mathit{lbound}$.

---

[2] To make $\Pi\Omega_k$ weaker than Π_k, we need to generalize the set $\mathcal{I}$ with a relation $\equiv$ as we did in splittable partitioned failure detectors Π_k^S in [6]. To keep the presentation focused on the main results, we omit this generalization.

[3] As a convention, $\max \emptyset = 0$.

We call a component that satisfies the liveness properties ($\Pi\Omega$2–5) a *live component*, and other components *non-live components*. Suppose for a live component P_j the *lbound* values eventually converge to ℓ_j. Intuitively, in live component P_j the failure detector behaves just like Ω_{ℓ_j},[4] the one known to solve ℓ_j-set agreement. The safety property ($\Pi\Omega$1) guarantees that the number of decisions from all components do not exceed k.

In general, the partition approach, proposed first in [6], is to partition the processes and require the safety properties to hold on all components while the liveness properties to hold only on at least one component. Failure detector class $\Pi\Omega_k$ can be viewed as applying static partitioning to Ω_k.[5]

The strength of $\Pi\Omega_k$ is fully characterized by Figure 1. We defer to Section 6 as a central place to study and compare the strength of all proposed failure detectors and avoid repetitions. We summarize the strength of $\Pi\Omega_k$ comparing with Ω_k and Υ in the following theorem.

Theorem 1 *The followings hold regarding the strength of* $\Pi\Omega_k$. *(1)* $\Pi\Omega_1 \equiv \Omega_1$. *(2)* $\Pi\Omega_k \prec \Omega_j$ *for all* $k \geq 2$, $j \geq 1$, *and* $k \geq j$. *(3)* $\Pi\Omega_k \not\preceq \Omega_j$ *and* $\Omega_j \not\preceq \Pi\Omega_k$ *for all* $k \geq 2$ *and* $k < j \leq n$. *(4)* $\Pi\Omega_k \prec \Pi\Omega_{k-1}$ *for all* $k \geq 2$. *(5)* $\Pi\Omega_k \not\preceq \Upsilon$ *and* $\Upsilon \not\preceq \Pi\Omega_k$, *for all* $k \geq 2$.

The key result is that $\Pi\Omega_k$ is incomparable with Υ for all $k \geq 2$. Therefore, $\Pi\Omega_k$ is a new class of failure detectors that is strictly weaker than Ω_k, but is strong enough to solve k-set agreement in shared-memory systems with arbitrary failure patterns. Together with Π_k and Π_k^S proposed in [6], these are the only classes known (to our best knowledge) to solve k-set agreement with arbitrary failure patterns and are strictly weaker than Ω_k.[6] Moreover, our results demonstrate that, even though Υ is very weak, we can still find a failure detector $\Pi\Omega_2$ to solve 2-set agreement, but $\Pi\Omega_2$ is not stronger than Υ.

3.2 Solving k-set agreement with $\Pi\Omega_k$

The algorithm using $\Pi\Omega_k$ to solve k-set agreement is based on an extension of the k–$converge$ algorithm presented in [18]. The original k–$converge$ algorithm forces every participant to use the same value of "k". With $\Pi\Omega_k$ failure detectors, we need processes in each component to try to converge on some decisions, the number of which is bounded by the *lbound* output of the failure detector. Therefore we extend the k–$converge$ algorithm by moving "k" into the parameter of the routine and rename the routine to $converge()$. We adjust the specification of $converge()$ as follows.

Routine $converge()$ takes in three parameters: ℓ is the upper bound on the number of values can be committed (this parameter corresponds to the "k" in k–$converge$), p is the process identifier, and v is the input value of the process. It outputs a pair (c, v'), where c is a boolean and v' is one of the input value. When p outputs (c, v'), we say that p picks v', and if $c =$ *True*, we say that p commits to v'. The routine satisfies the following properties: (1) C-Termination: Every correct process picks some value. (2) C-Validity: If a process p picks value v, then some process q invoked $converge()$ with parameter v. (3) C-Agreement: If a process p commits to a value, then at most ℓ_{max} values are picked, where ℓ_{max} is the maximum ℓ that processes pass into $converge()$. (4) Convergence: If all processes use the same value in the ℓ parameter ($\ell > 0$), and if there are no more than ℓ distinct input values, then every process that picks a value commits.

The first two properties are the same as in [18], while the last two properties are adjusted to accommodate different input values of ℓ. Although the interface and the specification are changed, the algorithm is exactly

[4] In [5] we show that a variation of failure detectors that output *isLeader* and *lbound*, named Ω_k'', is equivalent to Ω_k failure detectors.

[5] To be more exact, it is a static partitioning of Ω_k'' defined first in [5].

[6] The Υ^k failure detector proposed in [8] only solves k-set agreement in systems with at most k failures.

Shared variables:
  Register D, initially $\perp$
  $converge()$ instances: $converge[\,][\,]$

Output of failure detector $\Pi\Omega_k$ on process p_i:
  $isLeader_i$, $lbound_i$, cid_i

Code for process p_i:

```
1  v ← the input value of p_i
2  repeat
3     cid ← cid_i
4  until cid ≠ ⊥
5  r ← 0
6  repeat
7     c ← False
8     if isLeader_i = True then
9        r ← r + 1
10       (v, c) ← converge[cid][r](lbound_i, i, v)
11    if c = True then
12       D ← v; return (D)
13 until D ≠ ⊥
14 return (D)
```

Figure 2: k-set agreement algorithm using $\Pi\Omega_k$

the same as in [18], and the proof only needs some minor adjustment. We put the algorithm and the proof in the appendix for convenience.

Based on the $converge()$ routine, we provide an algorithm to solve k-set agreement using $\Pi\Omega_k$ in Figure 2. The algorithm is straightforward. We use *cid* output of failure detectors to isolate each component and make sure only processes in the same component could run the same instance of $converge()$ routine. Within a component, only those processes with *isLeader* output being *True* can run $converge()$ instances. Each $converge()$ instance only uses the output of the previous $converge()$ instance as the input, which is important to guarantee the safety of the algorithm. In any $converge()$ instance if some process p commits to a value v, then p writes v to a shared variable D and decides on v, and eventually all correct processes will see a non-$\perp$ D value and decide. The following theorem summarizes the correctness of the algorithm.

Theorem 2 *Algorithm in Figure 2 solves k-set agreement using failure detectors in $\Pi\Omega_k$, for any $k \geq 1$.*

Proof. It's obvious that k-set *Validity* holds.

For *Uniform k-Agreement*, we only need to consider decisions made in line 12, since decisions made in line 14 do not generate new decision values. Consider every component P_i. If some process decides in line 12, we consider the earliest such decision, say by a process $p \in P_i$. Process p decides v because it commits to v in an instance $converge[cid][r]()$. By the *C-Agreement* property of $converge()$, at most ℓ_{max} values can be picked in this $converge[cid][r]()$ instance, where ℓ_{max} is the maximum *lbound* values in the input of this instance. Since the algorithm guarantees for any $r' > r$, instances $converge[cid][r']()$ only uses the values picked in instance $converge[cid][r]()$, we know that there are at most ℓ_{max} values can be decided in line 12 by processes in component P_i. By definition, $\ell_{max} \leq lbound(P_i)$. Then, by property ($\Pi\Omega1$), there are at most k values that can be decided. So Uniform k-Agreement holds.

For k-set *Termination*, first by property ($\Pi C2$) all correct processes eventually exit the loop in lines 2–4. In the live component P_j that satisfies ($\Pi\Omega2$–5), eventually there is at least one correct process and at most ℓ processes in P_j invoking $converge()$, where ℓ is the eventually converged *lbound* output value. Moreover, all these processes invoke $converge()$ with the same first parameter value ℓ. Thus, the *C-Termination* and

Convergence properties guarantee that all correct processes in P_j eventually commit to some value in some $converge()$ instance. Therefore, eventually D is written. Once D is written, all correct processes eventually decide. □

4 Failure Detector $\Pi\Omega\Upsilon_k$

After defining $\Pi\Omega_k$, our next step is to find a mixture of $\Pi\Omega_k$ and Υ such that the new failure detectors are weaker than both and are still enough to solve n-set agreement. Since we know that $\Pi\Omega_k$ and Υ are not comparable, it immediately means that the new failure detectors are strictly weaker than both $\Pi\Omega_k$ and Υ. This leads us to the discovery of failure detectors $\Pi\Omega\Upsilon_k$.

4.1 Specification of $\Pi\Omega\Upsilon_k$

The output of $\Pi\Omega\Upsilon_k$ for process p is a tuple $(S, \mathit{lbound}, \mathit{cid})$, where S is a subset of P that informally matches the output of Υ, and *lbound* and *cid* outputs have the same value range and same informal meaning as the ones in $\Pi\Omega_k$. For a component P_j, let $\mathit{correct}(P_j) = \mathit{correct}(F) \cap P_j$, the set of correct processes in P_j (under a failure pattern F).

A failure detector $\mathcal{D}$ is in the class $\Pi\Omega\Upsilon_k$ if for any failure pattern F and any failure detector history $H \in \mathcal{D}(F)$, there exists a partition $\pi = \{P_1, \ldots, P_s\}$ of P, such that the following properties hold. The *cid* properties and safety properties are the same as $\Pi\Omega_k$, namely $(\Pi C1)$, $(\Pi C2)$, and $(\Pi\Omega 1)$. The liveness part specifies that there exists one partitioned component P_j such that $(\Pi\Omega 2)$ of $\Pi\Omega_k$ and the following property hold:

$(\Pi\Upsilon 1)$ P_j contains at least one correct process, and eventually all correct processes in P_j output the same $S \subseteq P_j$ such that S is not the set of correct processes in P_j and either $S \neq \emptyset$ or the number of correct processes is bounded by the eventual *lbound* output. Formally, $\mathit{correct}(P_j) \neq \emptyset \wedge \exists S_0 \subseteq P_j, S_0 \neq \mathit{correct}(P_j), \exists t_0, (\forall p \in \mathit{correct}(P_j), \forall t > t_0, (H(p,t).S = S_0 \wedge (S_0 \neq \emptyset \vee |\mathit{correct}(P_j)| \leq H(p,t).\mathit{lbound})))$.

We call a component that satisfies the liveness properties $(\Pi\Omega 2)$ and $(\Pi\Upsilon 1)$ a *live component*, and other components *non-live components*. Intuitively, in the live component P_j, the S output behaves almost the same as the output of Υ, except that S may eventually stabilize to $\emptyset$, in which case the number of correct processes in P_j must be bounded by the eventual *lbound* output. This mixture is important in making $\Pi\Omega\Upsilon_k$ strictly weaker than Υ. In particular, $\Pi\Omega\Upsilon_0$ is well-defined since *lbound* outputs could always be 0. However, in $\Pi\Omega\Upsilon_0$ the above mixture of requirements on S and on *lbound* is gone, and we will show that $\Pi\Omega\Upsilon_0$ is equivalent to Υ (the proof is not straightforward though).

The follow theorem summarizes the results on the strength of $\Pi\Omega\Upsilon_k$ comparing with $\Pi\Omega_k$ and Υ, which is captured in Figure 1 and will be studied in Section 6. The key result is that $\Pi\Omega\Upsilon_k$ is strictly weaker than Υ for any $k \geq 1$, and as k increases, its strength is strictly weakened. Therefore, we found a new family of n classes of failure detectors that are all strictly weaker than Υ. It now only shows that Υ is not the weakest failure detector ever, but also suggests that there are still plenty of room under Υ to fit in non-trivial failure detectors. The full proof of theorem 3 is left till Section 6.

Theorem 3 *The followings hold regarding the strength of* $\Pi\Omega\Upsilon_k$. *(1)* $\Pi\Omega\Upsilon_0 \equiv \Upsilon$. *(2)* $\Pi\Omega\Upsilon_k \prec \Pi\Omega\Upsilon_{k-1}$ *for all* $k \geq 1$. *(3)* $\Pi\Omega_j \not\preceq \Pi\Omega\Upsilon_k$ *for all* $1 \leq k \leq n$ *and* $1 \leq j \leq n$. *(4)* $\Pi\Omega\Upsilon_k \preceq \Pi\Omega_j$ *for all* $k \geq j \geq 1$. *(5)* $\Pi\Omega\Upsilon_k \not\preceq \Pi\Omega_j$ *for all* $j \geq k+2$ *and* $k \geq 1$.

Shared variables:
  Registers D, $D[\,][\,]$, initially $\perp$
  Registers $M[1\ldots n+1]$, initially $\perp$
  Binary registers $Stable[\,][\,]$, initially *True*
  $converge()$ instances: $converge$, $converge[\,][\,]$, $converge[\,][\,][\,]$

Output of failure detector $\Pi\Omega\Upsilon_k$ on process p_i:
  S_i, *lbound*$_i$, *cid*$_i$

Code for process p_i:

1 $v \leftarrow$ the input value of p_i
2 $r \leftarrow 0$
3 **repeat**
4   $cid \leftarrow cid_i$
5 **until** $cid \neq \perp$
6 $M[i] \leftarrow cid$
7 $(v, c) \leftarrow converge(n, i, v)$
8 **if** $c = True$ **then**
9   $D \leftarrow v$; **return** (D)
10 $m \leftarrow |\{j | M[j] = cid\}|$
11 **repeat**
12   $r \leftarrow r + 1$
13   $k' \leftarrow \max(m-1, lbound_i)$
14   $(v, c) \leftarrow converge[cid][r](k', i, v)$
15   **if** $c = True$ **then**
16     $D \leftarrow v$; **return** (D)
17   $S \leftarrow S_i$
18   **if** $p_i \notin S$ **then** $D[cid][r] \leftarrow v$
19   **else**
20     $r' \leftarrow 0$
21   **repeat**
22       $r' \leftarrow r' + 1$
23       $(v, c) \leftarrow converge[cid][r][r'](|S| - 1, i, v)$
24   **if** $c = True$ **then** $D[cid][r] \leftarrow v$
25   **if** $S \neq S_i$ **then** $Stable[cid][r] \leftarrow False$
26   **until** $D \neq \perp$ **or** $D[cid][r] \neq \perp$ **or** $\neg Stable[cid][r]$
27   **if** $D[cid][r] \neq \emptyset$ **then** $v \leftarrow D[cid][r]$
28 **until** $D \neq \perp$
29 **return** (D)

Figure 3: n-set agreement algorithm using $\Pi\Omega\Upsilon_k$

4.2 Solving n-set agreement with $\Pi\Omega\Upsilon_k$

In Figure 3, we show an n-set agreement algorithm using the $\Pi\Omega\Upsilon_k$ failure detectors. The algorithm is based on the n-set agreement algorithm using Υ in [8] with a few important modifications.

The main repeat-until loop (lines 11–28) is almost the same as the algorithm in [8] except two modifications. First, we use *cid* output to isolate every component by using different $coverge[cid]$ instances and different set of $D[cid][\,]$ and $Stable[cid][\,]$ variables. Second, in lines13–14, each process p_i needs to determine an appropriate converge number k' for the $converge[cid][r]$ instance for its own component. Let P_i be the component containing p_i and let $m = |P_i|$. If we follow exactly like in the algorithm of [8], then we should set $k' = m - 1$. However, to use $\Pi\Omega\Upsilon_k$, we need to set $k' = \max(m-1, lbound_i)$ (line 13), where *lbound*$_i$ is the *lbound* output on p_i. This is to cover the case where the S output in the component stabilizes to $\emptyset$ and all processes in P_i are correct. In this case, if k' keeps to be $m-1$, processes will repeat forever in the loop and no process can commit to a value and decide. However, in this case we know that *lbound* eventually converge to a value that is at least m, so we use $k' = m$ in this case to guarantee that processes in P_j can commit to a value and decide.

The next question is how p_i could get m. The new code in lines 3–10 is for this purpose. Process p_i first waits for long enough time to get a non-$\perp$ *cid* output. Then p_i record its *cid* in $M[i]$. Next p_i runs a global $converge()$ instance with n as the converge number. The purpose of this $converge()$ is that, if some process crashes before running this instance, then correct processes commit to some values and they can already decide. If no one can decide after this instance, then it must be the case that all processes have invoked this instance, which means all processes have recorded their *cid*'s into $M[\,]$. Thus $M[\,]$ contains the complete partition information and p_i can obtain the size of its component from $M[\,]$ (line 10). Note that, to guarantee safety, after this $converge()$ instance all processes should only use the output values of the instance in the later algorithm.

The rest of the algorithm is exactly the same as the one in [8]. Essentially, processes in each component repeatedly invoke new $converge()$ instances to try to commit a value. In a live component P_j, the S output eventually stabilizes at a value $S_0 \subseteq P_j$ that is not the set of correct processes in P_j. So either processes in S_0 eliminate one value in the inner loop (lines 21–26) since S_0 contains some crashed process, or they choose a value recorded by some correct process outside S_0 (line 18). The only exception is $S_0 = \emptyset$ and all processes in P_j are correct, but this is covered by line 13, in which *lbound* $\geq m$ will be chosen. We refer to [8] for further details about the algorithm.

The Uniform k-Agreement is satisfied, because (a) if some process decides after the global $converge()$, at most n values are left for the rest of the algorithm; and (b) if no process decides after the global $converge()$, each component P_i may generate at most $\max(|P_i| - 1, \textit{lbound}(P_i))$ decisions, and by ($\Pi\Omega 1$) there are at most n decisions. So we have

Theorem 4 *The algorithm in Figure 3 solves n-set agreement using $\Pi\Omega\Upsilon_k$ for any $k \geq 0$.*

Proof. We consider an arbitrary execution of the algorithm. *Validity* comes directly from our algorithm and the *Validity* property of the $converge$ algorithm.

For *Uniform n-Agreement*, a process either follows other processes' decision, or makes an original decision at line 9 or 16. If there is a process commits to a value at line 7, the *C-Agreement* property ensures that the number of distinct values carried by the processes in the statements in line 8–29 would not be larger than n. So *Uniform n-Agreement* holds. Suppose nobody commits at line 7, thus nobody decides at line 9. Because the processes in different components are isolated by *cid*'s when executing line 11–29, we can consider the number of original decisions in each component separately. Suppose processes in component P_j make n_j original decisions. Because the original decisions must pass the $converge$ check at line 14, we know that $n_j \leq k'_j$ where k'_j is calculated at line 13. Therefore $n_j \leq \textit{lbound}(P_j)$ or $n_j \leq |P_j| - 1$ (because m is the number of processes whose *cid*'s already appear in $M[\,]$, $m \leq |P_j|$). If $n_j \leq |P_j| - 1$, because of component isolation, we know that at least one process's input value is discarded. Since there are totally $n+1$ processes in the system, the number of original decisions is no larger than n. So *Uniform n-Agreement* holds. If for all j, $n_j > |P_j| - 1$, we have $n_j \leq \textit{lbound}(P_j)$. According to ($\Pi\Omega 1$), $\sum_{j=1}^{s} \textit{lbound}(P_j) \leq k \leq n$. *Uniform n-Agreement* still holds.

For *Termination*, we assume nobody decides to reach a contradiction, since our algorithm ensures that every correct process decide as long as one process decides. First, no correct process would be blocked in the loop of waiting non-$\perp$ *cid* output (lines 3–5), according to ($\Pi C1$). So all correct processes enter the repeat-until loop in lines 11–28 but never leave it.

We now consider the membership array M. According to the *C-Termination* property, each of the correct process in P_j must pick a value at line 7. Since none of them decides at line 9, they must not commit to the picked value. According to the *Convergence* property of $converge()$, there must be at least $n + 1$ distinct input values to this $converge()$ instance at line 7. We claim that all $n + 1$ processes must have invoked this

$converge()$ by the time the first process get the (*False*, $-$) return value from the instance. If not, processes not invoking the instance could have crashed and never invoke the instance. Then by the *Convergence* property, the first process should commit to a value. Therefore, we know that all processes in must have recorded their non-$\perp$ *cid* in array M by the time the first process returns from the $converge()$ at line 7. So at line 10, the m calculated is the exact size of the process's component.

Consider a live component P_j. Let t_1 be the time after which all processes in $P_j \setminus \mathit{correct}(P_j)$ have already crashed, and the *lbound* and S output of $\Pi\Omega\Upsilon_k$ become stable for the correct processes. According to ($\Pi\Omega 2$), the *lbound* outputs of all the correct processes are the same. Let it be ℓ_j. According to ($\Pi\Upsilon 1$), the S outputs of all the correct processes have the same value S_0 and $S_0 \neq \mathit{correct}(P_j)$. Let t_2 be the time after which all correct processes have announced their *cid*'s at line 6. This means none of the cells in the shared array M will be updated after $t = \max(t_1, t_2)$.

After time t, every process in P_j will use the same stable k' calculated at line 13 to reach agreement. Before S output stabilizes to S_0, no process is stuck in the inner loop (line 21–26) due to the $Stable[\,][\,]$ variable. After S stabilizes to S_0, because $S_0 \subseteq P_j$ and $S_0 \neq \mathit{correct}(P_j)$, either a correct process is in $P_j \setminus S_0$ or S_0 contains a crashed process. So no process will be stuck in the inner loop (line 21–26). Since no process in P_j decides at line 16, it must be the case that $\ell_j < |\mathit{correct}(P_j)|$ (otherwise after t at most $\ell_j \leq k'$ values can be invoked in $converge()$ at line 14 and processes should decide). By property ($\Pi\Upsilon 1$), it implies that $S_0 \neq emptyset$. Moreover, we have $k' = m - 1$. This means all processes in P_j must be correct, since otherwise at most $m-1$ values can be invoked in $converge()$ at line 14. In this case, no process in the inner loop $converge()$ (line 23) can commit, so they will wait for the values picked by processes in $P_j \setminus S_0$ at line 18. Since $S_0 \neq \emptyset$, at most $m - 1$ values can be picked. Therefore, in the next $converge()$ instance at line 14, all processes will decide — a contradiction. So *Termination* also holds. □

5 Failure Detector $\Pi\Upsilon$

To further demonstrate the power of partition approach, in this section, we directly apply the approach to failure detector Υ and define a failure detector $\Pi\Upsilon$ that is weaker than Υ but is still strong enough to solve n-set agreement.

5.1 Specification of $\Pi\Upsilon$

The output of $\Pi\Upsilon$ for process p is a tuple (S, cid), where S is a subset of P that is supposed to match the Υ output, *cid* has the same value range and meaning as in $\Pi\Omega_k$ and $\Pi\Omega\Upsilon_k$. For the component ID set $\mathcal{I}$, we further require that $\mathcal{I}$ has a total order $\leq$ among all *cid* values.

A failure detector $\mathcal{D}$ is in the class $\Pi\Upsilon$ if for any failure pattern F and any failure detector history $H \in \mathcal{D}(F)$, there exists a partition $\pi = \{P_1, \ldots, P_s\}$ of P, such that the following properties hold. First, *cid* output satisfies the same properties ($\Pi C1$) and ($\Pi C2$) as in $\Pi\Omega_k$. Second, ***one of the following properties*** hold:

($\Pi C3$) There exist two components that both contain correct processes. Formally, $\exists P_i, P_j, \mathit{correct}(P_i) \neq \emptyset \wedge \mathit{correct}(P_j) \neq \emptyset$.

($\Pi\Upsilon 2$) There exists one component P_j with at least one correct process, such that eventually all correct processes in P_j output the same nonempty set $S \subseteq P_j$, and S is not the set of correct processes in P_j. Formally, $\exists P_j, \mathit{correct}(P_j) \neq \emptyset \wedge \exists t_0 \in \mathcal{T}, \exists S_0 \subseteq P_j, S_0 \neq \emptyset \wedge S_0 \neq \mathit{correct}(P_j) \wedge (\forall p \in \mathit{correct}(P_j), \forall t > t_0, H(p,t).S = S_0)$.

Note that ($\Pi\Upsilon 2$) implies that the component P_j in the property has at least two processes. The inclusion of ($\Pi C3$) as an alternative to ($\Pi\Upsilon 2$) is important to make $\Pi\Upsilon$ weaker than Υ. It is easy to see that, if we would remove ($\Pi C3$), $\Pi\Upsilon$ would be equivalent to $\Pi\Omega\Upsilon_0$, which we show to be equivalent to Υ. The following theorem summarizes the strength of $\Pi\Upsilon$ comparing with $\Pi\Omega_k$, $\Pi\Omega\Upsilon_k$ and Υ.

Theorem 5 *The followings hold regarding the strength of* $\Pi\Omega\Upsilon_k$. *(1)* $\Pi\Upsilon \prec \Upsilon$ *when* $n \geq 3$, *and* $\Pi\Upsilon \equiv \Upsilon$ *when* $n \leq 2$. *(2)* $\Pi\Upsilon \not\preceq \Pi\Omega_k$ *and* $\Pi\Omega_k \not\preceq \Pi\Upsilon$ *for all* $k \geq 2$. *(3)* $\Pi\Upsilon \not\preceq \Pi\Omega\Upsilon_k$ *for all* $k \geq 1$. *(4)* $\Pi\Omega\Upsilon_{n-1} \preceq \Pi\Upsilon$. *(5)* $\Pi\Omega\Upsilon_k \not\preceq \Pi\Upsilon$ *for all* $1 \leq k \leq n-3$, *or* $k = n-2$ *and* n *is odd.*

The full proof of theorem 5 is left till Section 6. The key result is that $\Pi\Upsilon$ is strictly weaker than Υ. Thus by direct application of the partition approach, we also find a new class of failure detectors weaker than Υ. More interestingly, we find $\Pi\Upsilon$ to be incomparable with $\Pi\Omega\Upsilon_k$ when $1 \leq k \leq n-3$ (also when $k = 2$ and n is odd), but $\Pi\Upsilon$ is strictly stronger than $\Pi\Omega\Upsilon_{n-1}$. It hints that even though Υ is very weak, there are still multiple ways to weaken it and discover different kind of weaker failure detectors.

5.2 Solving n-set agreement with $\Pi\Upsilon$

The basic idea is for each component P_i to run a Υ-based $(|P_i| - 1)$-set agreement algorithm, where $|P_i|$ is obtained in the same way as in the algorithm of Figure 3. If ($\Pi\Upsilon 2$) is satisfied on a component P_j, then P_j eliminates one value and achieves $(|P_j| - 1)$-set agreement, which also means that globally one value is eliminated and n-set agreement is accomplished. Otherwise, ($\Pi C3$) is satisfied, in which case, a component with a larger *cid* must eventually see a value from a component with a smaller *cid* and the former can immediately decides on the value seen, because the total order of *cid*'s guarantee that at least one value is eliminated in the component with the largest *cid*.

Figure 4 shows the n-set agreement algorithm using $\Pi\Upsilon$.

Theorem 6 *The algorithm in Figure 4 solves* n*-set agreement using* $\Pi\Upsilon$.

Proof. We consider an arbitrary execution of the algorithm. *Validity* comes directly from our algorithm and the *C-Validity* property of the $converge()$ routine.

For *Uniform* n*-Agreement*, if there is a process that commits to a value at line 7, the *C-Agreement* property ensures that the number of distinct values carried by the processes in the statements in line 8–35 would not be larger than n. So *Uniform* n*-Agreement* holds in this case.

Suppose no process commits to a value at line 7. In this case, in the rest of algorithm processes all use values in the array $V[\,]$. If $V[\,]$ contains at most n distinct values, *Uniform k-Agreement* already holds. So suppose that $V[\,]$ contains $n+1$ distinct values. In this case, by the same argument as in the proof of Theorem 4, we know that by the time p_i executes line 11, array $M[\,]$ contains only non-$\perp$ *cid* values, and the m computed is the exact size of the component that p_i belongs to. We use P_j*.cid* to denote the non-$\perp$ *cid* value in the output of processes in P_j.

We consider the component P_j with the largest P_j*.cid*, based on the total order $\leq$ among the *cid* values. Because P_j*.cid* is the largest, any process in any component that decides at line 15 or 27 must decide on a value not from P_j. Moreover, any process in other components that decides at line 19 can only decide a value from its own component. Therefore, only processes in P_j can decide a $V[\,]$ value belonging to P_j at line 19. If none of the process in P_j ever decides at line 19, then at least one value in $V[\,]$ belonging to P_j is not a decision value, so only n values could be decision values. If some process in P_j decides at line 19,

Shared variables:
Registers D, $D[\,][\,]$, initially $\perp$
Registers $M[1 \ldots n+1]$, initially $\perp$
Registers $V[1 \ldots n+1]$, initially $\perp$
Binary registers $Stable[\,][\,]$, initially *True*
$converge()$ instances: $converge$, $converge[\,][\,]$, $converge[\,][\,][\,]$

Output of failure detector $\Pi\Upsilon$ on process p_i:
S_i, cid_i

Code for process p_i:

1 $v \leftarrow$ the input value of p_i
2 $r \leftarrow 0$
3 **repeat**
4 $\quad cid \leftarrow cid_i$
5 **until** $cid \neq \perp$
6 $M[i] \leftarrow cid$
7 $(v, c) \leftarrow converge(n, i, v)$
8 $V[i] \leftarrow v$
9 **if** $c = True$ **then**
10 $\quad D \leftarrow v$; **return** (D)
11 $m \leftarrow |\{j | M[j] = cid\}|$
12 **repeat**
13 $\quad \mathcal{V} \leftarrow \{V[j] \mid M[j] \leq cid \wedge M[j] \neq cid \wedge V[j] \neq \perp\}$
14 $\quad$ **if** $\mathcal{V} \neq \emptyset$ **then**
15 $\quad\quad D \leftarrow$ arbitrary element in $\mathcal{V}$; **return** (D)
16 $\quad r \leftarrow r + 1$
17 $\quad (v, c) \leftarrow converge[cid][r](m-1, i, v)$
18 $\quad$ **if** $c = True$ **then**
19 $\quad\quad D \leftarrow v$; **return** (D)
20 $\quad S \leftarrow S_i$
21 $\quad$ **if** $p_i \notin S$ **then** $D[cid][r] \leftarrow v$
22 $\quad$ **else**
23 $\quad\quad r' \leftarrow 0$
24 $\quad\quad$ **repeat**
25 $\quad\quad\quad \mathcal{V} \leftarrow \{V[j] \mid M[j] \leq cid \wedge M[j] \neq cid \wedge V[j] \neq \perp\}$
26 $\quad\quad\quad$ **if** $\mathcal{V} \neq \emptyset$ **then**
27 $\quad\quad\quad\quad D \leftarrow$ arbitrary element in $\mathcal{V}$; **return** (D)
28 $\quad\quad\quad r' \leftarrow r' + 1$
29 $\quad\quad\quad (v, c) \leftarrow converge[cid][r][r'](|S| - 1, i, v)$
30 $\quad\quad\quad$ **if** $c = True$ **then** $D[cid][r] \leftarrow v$
31 $\quad\quad\quad$ **if** $S \neq S_i$ **then** $Stable[cid][r] \leftarrow False$
32 $\quad\quad$ **until** $D \neq \perp$ **or** $D[cid][r] \neq \perp$ **or** $\neg Stable[cid][r]$
33 $\quad\quad$ **if** $D[cid][r] \neq \perp$ **then** $v \leftarrow D[cid][r]$
34 **until** $D \neq \perp$
35 **return** (D)

Figure 4: n-set agreement algorithm using $\Pi\Upsilon$

we know that there will be at most $|P_j| - 1$ distinct values belonging to P_j that can be decided at line 19 by *C-Agreement* of $converge()$. Therefore, in both cases, *Uniform n-Agreement* holds.

For *Termination*, we only need to prove that at least one process will decide and write the register D, since as long as D is written every correct process eventually decides. We assume for a contradiction that no process decides. First, ($\Pi C1$) ensures that no correct process be blocked at lines 3–5, waiting for the non-$\perp$ cid. Second, by the same argument as in the proof of Theorem 4, when a process p enters the repeat-until loop (lines 12–34), its m value is the size of its component.

According to the definition of $\Pi\Upsilon$, we consider two cases: 1) ($\Pi C3$) holds, and 2) ($\Pi\Upsilon 2$) holds.

In case 1), suppose P_i, P_j are two components such that $correct(P_i) \neq \emptyset$ and $correct(P_j) \neq \emptyset$. Let $p \in correct(P_i)$, $q \in correct(P_j)$. Without loss of generality, we assume $P_i.cid \leq P_j.cid$. Since both

p and q does not made decision at line 10, eventually both of them write their values into the V array. So eventually, q will read p's value and decides either at line 15 in the outer repeat-until loop, or at line 27 in the inner repeat-until loop. *Termination* holds in this case.

In case 2), suppose P_j is the component satisfying $(\Pi\Upsilon 2)$. If nobody makes any decision at line 10, 15 or 27, then processes in P_j runs the algorithm isolated from other components with their S output exactly like the processes would run the algorithm using the Υ failure detector. Therefore, *Termination* also holds in this case. □

6 Comparing failure detectors

This section is the central place to show all the results captured in Figure 1 and stated in Theorems 1, 3, and 5. Since Υ is already a very weak failure detector, one can imagine that show that under Υ there are still such structure in which a series of failure detectors have various strength would be a subtle and delicate task. Indeed, besides those obvious transformations, other results on possible or impossible transformations are quite delicate and require subtle techniques to prove them (and a few of them are still open). These proofs really show the subtle relationship between the failure detectors. Unfortunately, due to the space constraint, we have to move the full proofs of impossible transformations into appendix. To compensate, we provide intuitive ideas and proof outlines for those key proofs.

6.1 Possible transformations

For possible transformations, we need to prove all the arrows in Figure 1. Most transformations are obvious from the failure detector definitions.

Lemma 1 *(1)* $\Pi\Omega_k \preceq \Pi\Omega_{k-1}$*; (2)* $\Pi\Omega\Upsilon_k \preceq \Pi\Omega\Upsilon_{k-1}$*; (3)* $\Pi\Omega_k \preceq \Omega_k$*; (4)* $\Pi\Omega\Upsilon_k \preceq \Upsilon$*; (5)* $\Pi\Upsilon \preceq \Upsilon$.

Proof. The first two parts hold directly by the definition of the failure detectors. The last three parts hold because we can treat Ω_k and Υ as a special case of partitioned failure detectors with only a single component P. □

A few of the transformations need extra explanations.

Lemma 2 $\Pi\Omega_1 \equiv \Omega_1$.

Proof. In the paper [5], we show that Ω_k is equivalent to Ω''_k, so in the following proof, we use Ω''_1 instead of Ω_1. In Ω''_k, the failure detector output is $(\mathit{isLeader}, \mathit{lbound})$, where *isLeader* is a boolean and *lbound* is a non-negative integer of at most k. It requires that eventually the output on each process stabilizes, and the *lbound* on all processes are the same, and there is at least one and at most ℓ eventual leaders where ℓ is the stabilized *lbound* value.

By Lemma 1, we only need to show $\Omega''_1 \preceq \Pi\Omega_1$. We construct a transformation from $\Pi\Omega_1$ to Ω''_1 as follows. Let $(\mathit{isLeader}, \mathit{lbound}, \mathit{cid})$ be the output of failure detector in $\Pi\Omega_1$ and $(\mathit{isLeader}', \mathit{lbound}')$ be output of failure detector in Ω''_1. We set lbound' to 1 on all processes. For process p, we set $\mathit{isLeader}'$ to be *False* if $p.\mathit{lbound} = 0$ and to be *isLeader* if $p.\mathit{lbound} = 1$. By the definition of $\Pi\Omega_1$, there are exact one component P_j with $\mathit{lbound} = 1$ ever, so the $\mathit{isLeader}'$ outputs of processes in all other components are always *False*. Component P_j is the only live component and eventually, there are exact one correct process be the leader in P_j. Therefore, there are exact one correct process which set $\mathit{isLeader}' = \mathit{True}$ in the output eventually. This gives the Ω''_1 failure detector. □

Shared variables:
  Registers $L[\,]$, the value is a tuple $(cid, isLeader, r)$,
    initially $(\perp, False, 0)$

Output of failure detector $\Pi\Omega_k$ on process p_i:
  $(cid_i, lbound_i, isLeader_i)$
Output of failure detector $\Pi\Omega\Upsilon_k$ on process p_i:
  $(S'_i, cid'_i, lbound'_i)$, initially $(\emptyset, \perp, 0)$

Local variables on process p_i:
  cid
  isLeader
  r, round number
  C, estimated membership of component containing p_i
  A, leaders
  B, *lbound* leaders with highest r

Code for process p_i:
1 **repeat**
2   $cid \leftarrow cid_i$
3 **until** $cid \neq \perp$
4 $cid'_i \leftarrow cid$
5 $r \leftarrow 0$
6 **repeat forever**
7   $isLeader = isLeader_i$
8   $lbound'_i = lbound_i$
9   **if** $isLeader = True$ **then**
10     $r \leftarrow r + 1$
11   $L[i] \leftarrow (cid, isLeader, r)$
12   $C \leftarrow \{j | L[j].cid \neq \perp \wedge L[j].cid = cid\}$
13   $A \leftarrow \{j | j \in C \wedge L[j].isLeader = True\}$
14   $B \leftarrow$ a subset of A such that $|B| \leq lbound'_i$ and
     $\forall j \in B, j' \in A \setminus B, L[j].r > L[j'].r$
     $\vee (L[j].r = L[j'].r \wedge j > j')$
15   $S'_i \leftarrow \{p_j | j \in C \setminus B\}$

Figure 5: Transform $\Pi\Omega_k$ into $\Pi\Omega\Upsilon_k$

Lemma 3 $\Pi\Omega\Upsilon_k \preceq \Pi\Omega_k$ *for all* $k \geq 1$.

Proof. Figure 5 provides a transformation from $\Pi\Omega_k$ to $\Pi\Omega\Upsilon_k$. The idea is for each component to come up with the set of at most $lbound$ leaders, then the S output of $\Pi\Omega\Upsilon_k$ is the complement of the leader set with respect to the component, and *lbound* and *cid* outputs of $\Pi\Omega\Upsilon_k$ are copied from $\Pi\Omega_k$. The key is that for a live component, S output eventually stabilizes to a set that cannot be the set of correct process (because at least one correct leader process is not in S), and if S is $\emptyset$, the *lbound* must be at least the number of correct processes in the component.

More specifically, to get the output for S, we introduce an array of shared registers L. Every process p_i waits to read a non-$\perp$ *cid* output (lines 1–3 in Figure 5). Then it periodically writes the non-$\perp$ *cid*, and the *isLeader* output from its $\Pi\Omega_k$ detector and an increasing round number r into $L[i]$ (line 11). Every process p_i also periodically scans the array L to compute C, the estimate the membership P_i of its own component, and find out the "leaders" A in its component (lines 12, 13). Let B to be the *lbound* processes in A with the highest $\langle r, i\rangle$ values (line 14). Then $S = C \setminus B$ (line 15).

For correctness, we consider a live component P_j with respect to $\Pi\Omega_k$. It is obvious that B is actually the Ω_l output within the P_j, where $l = lbound(P_j)$. Let C_j be the eventually stabilized value of C computed in line 12 for all correct processes in P_j. So as long as C_j is larger than B, S is not empty and thus ($\Pi\Upsilon 1$) is satisfied. If C_j is equal to B, let ℓ_j be the eventually stabilized *lbound* output for processes in P_j. Since

Shared variables:
  Registers $M[1..n+1]$, initially $\perp$
  Registers $N[]$: the value is a tuple (S, cnt), initially $(\emptyset, 0)$

Output of failure detector $\Pi\Omega\Upsilon_0$ on process p_i: $(S_i, \mathit{lbound}_i, \mathit{cid}_i)$
Output of failure detector Υ on process p_i: S'_i, initially P

Local variables on process p_i:
  $P[]$, initially $\emptyset$
  $count$, initially 1
  C,initially $\emptyset$

Code for process p_i:
1 **repeat**
2   **if** $M[i] = \perp$ **then** $M[i] \leftarrow \mathit{cid}_i$
3   $S'_i \leftarrow \{p_i | M[i] = \perp\}$
4 **until** $S'_i = \emptyset$
5 $\mathit{cid} \leftarrow M[i]$
6 $S'_i \leftarrow P$
7 $C \leftarrow \{M[i] | 1 \le i \le n+1\}$
8 **for** $j \in C$ **do** $P[j] \leftarrow \{i | M[i] = j\}$
9 $N[\mathit{cid}] \leftarrow (S_i, count)$
10 **repeat forever**
11   **if** $N[\mathit{cid}].S \neq S_i$ **then**
12     $count \leftarrow N[\mathit{cid}].cnt + 1$
13     $N[\mathit{cid}] \leftarrow (S_i, count)$
14   choose $j \in C$ with minimum $N[j].cnt$ such that
15   (1)$N[j].S \neq \emptyset$ and (2)$N[j].S \subseteq P[j]$
16   **if** such j exists **then** $S'_i \leftarrow N[j].S$
17   **else** $S'_i \leftarrow P$

Figure 6: Transform $\Pi\Omega\Upsilon_0$ into Υ

$|B| \le \ell_j$, and all correct processes in P_j are contained in C_j, we know that the number of correct processes is at most ℓ_j. In this case, S' output can be empty and ($\Pi\Upsilon 1$) is still satisfied. $\square$

Lemma 4 $\Pi\Omega\Upsilon_0 \equiv \Upsilon$.

Proof. $\Pi\Omega\Upsilon_0 \preceq \Upsilon$ is because a failure detector in Υ can be viewed as a failure detector in $\Pi\Omega\Upsilon_0$ with the full process set P as the single component and $\mathit{lbound} = 0$ for all processes.

Then, we present a transformation algorithm from a failure detector $\mathcal{D}$ in $\Pi\Omega\Upsilon_0$ into a failure detector $\mathcal{D}'$ in Υ in Figure 6. Process p_i first waits to see valid *cid* outputs of all processes (line 1 - 4). If some process p_j crashes before it completes writing cid_j into $M[j]$, S' outputs of all correct processes converge to the set of such processes (line 3) which contain only crash processes. Otherwise, all correct processes successfully learn the partition of $\Pi\Omega\Upsilon_0$ and record it in array $P[\,]$ (line 8). Then, p_i uses array $N[\,]$ to compute the S' output. $N[\mathit{cid}].S$ represents the latest S output in component $P[\mathit{cid}]$ while $N[\mathit{cid}].\mathit{count}$ represents the number of times that processes in component $P[\mathit{cid}]$ find S output is not stable. Eventually S' output is stable because the live component with respect to $\Pi\Omega\Upsilon_0$ satisfies the conditions in line 15 and its count eventually stops increasing. Since any stable S output satisfying conditions in line 15 satisfies the property of Υ, we can get a correct S' output by the algorithm.

Formally, there are two cases we consider.

Case 1: No process leaves the first repeat-until loop (lines 1–4). Eventually, every correct process eventually has the stable *cid* output (by $(\Pi C1)$) and every non-correct process does not change $M[i]$. So, S' outputs of all correct processes are the same and stable and non-empty eventually. This eventual S' output cannot be the exact set of all correct processes since at least one correct process has $\mathit{cid} \neq \perp$ eventually.

Case 2: if there exists a process leaving the first repeat-until loop. This means $M[i] \neq \perp$ for $1 \leq i \leq n+1$ at some time. By line 2, we know that once $M[i]$ is set to non-$\perp$ value, it remains to be non-$\perp$, so all correct processes eventually leave the first repeat-until loop, and all correct processes have the same local variable C and P_j with $\cup_{j \in C} P_j = P$ in line 8. Let $P_j.cid$ denote the *cid* corresponding to component P_j. Consider a live component P_j in the output of $\Pi\Omega\Upsilon_0$, then P_j contains at least one correct process and $lbound(P_j) = 0$ by $(\Pi\Omega1)$. So, by $(\Pi\Upsilon1)$, eventually all correct processes in P_j output the same $S \subseteq P_j$ such that S is not the set of correct processes in P_j and $S \neq \emptyset$. Then, eventually, $N[P_j.cid]$ satisfies all conditions in line 15 and never changes any more. If for some component P_i, $N[P_i.cid].S$ changes infinite often, by lines 12–13, $N[P_i.cid].cnt$ will eventually exceed $N[P_j.cid].cnt$. Therefore, by lines 14–16, S' outputs of all correct processes are stable and the same eventually.

Suppose, for a contradiction that the eventual S' outputs of all correct processes is the exact set of correct processes. Suppose $S' = N[P_i.cid].S$ for some component P_i. Then any process $p \notin S'$ crashes. Since $N[P_i.cid].S \subseteq P_i$ by line 15, only component P_i contains correct processes. So, P_i is the live component according to the output of $\Pi\Omega\Upsilon_0$. Then, the eventual S output of $\Pi\Omega\Upsilon_0$ of all correct processes in P_i is not the set of correct processes in P_i which is also not the set of correct processes in P. But eventually, $S' = N[P_i.cid].S = S$, the stabilized S output of $\Pi\Omega\Upsilon_0$ in P_i. This is a contradiction. So, we know the eventual S' outputs of all correct processes is not the exact set of correct processes, which completes our proof. □

Lemma 5 $\Pi\Omega\Upsilon_{n-1} \preceq \Pi\Upsilon$.

Proof. In Figure 7 we present an algorithm that transforms the output of a $\Pi\Upsilon$ failure detector to one of a $\Pi\Omega\Upsilon_{n-1}$ detector. Process p_i first waits for a valid *cid* output from $\Pi\Upsilon$, and then does an n-converge using its *cid* as the value. If it does not commit to a value, it then uses an n-agreement algorithm based on $\Pi\Upsilon$ to pick a *cid*. The purpose of this code is to guarantee that (a) if the $\Pi\Upsilon$ has $n+1$ distinct components, then the generated $\Pi\Omega\Upsilon_{n-1}$ has at most n components, and (b) if the $\Pi\Upsilon$ has at most n distinct components, then the generated $\Pi\Omega\Upsilon_{n-1}$ output has exactly the same components as those in $\Pi\Upsilon$. We will make it clear why it is so and why we need this property shortly. After announcing the picked *cid* into shared array N, the process starts to provide the $\Pi\Omega\Upsilon_{n-1}$ outputs. It's obvious that $(\Pi C1)$ and $(\Pi C2)$ are satisfied, because the processes will not change their cid^{out} as long as the value is set.

If there are some processes crashed before announcing their picked *cid* into shared array N, then eventually every process's $lbound^{out}$ is set to 0, and S^{out} is set to these crashed processes. (lines 9–13). Because every process set its $lbound^{out}$ to 0, $(\Pi\Omega1)$ and $(\Pi\Omega2)$ are satisfied. Let the set of crashed processes not announcing their picked *cid* in N be P_f. Since processes in P_f do not assign their cid^{out} yet, for any failure pattern we can always find a partition π which contains a component P_j such that P_j contains at least one correct process and $P_f \subset P_j$. So P_j is a live component in the sense of $(\Pi\Upsilon1)$.

If all processes announced their picked *cid*'s in N, every correct process eventually exits the loop at lines 11–13. In this case every process is able to derive a partition scheme that covers all $n+1$ processes from N (line 14). Because the number of distinct *cid*'s picked by the processes is at most n, there must exist a component which contains at least 2 processes. So the component C calculated at line 16 contains at least two processes. Then every process not in C sets its $lbound^{out}$ to its component size and its S^{out} to $\emptyset$, and processes in C set their $lbound^{out}$ to 0. Because $|C| \geq 2$, $(\Pi\Omega1)$ still holds. Now all components other than C appears to be live components in the sense of $(\Pi\Omega2)$ and $(\Pi\Upsilon1)$, because all processes set their $lbound^{out}$ and S^{out} to their component size and $\emptyset$, respectively. If there is a component $P_j \neq C$ that contains a correct process, then we are set.

```
Shared variables:
    Registers M[1 ... n+1], initially ⊥
    Registers N[1 ... n+1], initially ⊥
    Convergence instance: converge
    n-set agreement instance: n–agreement

Output of failure detector ΠΥ on process p_i:
    (S_i^in, cid_i^in)
Output of failure detector ΠΩΥ_{n-1} on process p_i:
    (S_i^out, cid_i^out, lbound_i^out), initially (∅, ⊥, 0)

Code for process p_i:
1  repeat
2    cid ← cid_i^in
3  until cid ≠ ⊥
4  M[i] ← cid
5  (cid, c) ← converge(n, i, cid)
6  if c ≠ True then
7    cid ← n–agreement(cid)
8  N[i] ← cid
9  cid_i^out ← cid
10 lbound_i^out ← 0
11 repeat
12   S_i^out ← {p_j | N[j] = ⊥}
13 until S_i^out = ∅
14 π ← partition derived from N
15 m ← max({|P_j| | P_j ∈ π}) //m ≥ 2
16 C ← a component in π such that |C| = m and C has
     the smallest cid among the same size components
17 if p_i ∉ C then
18   lbound_i^out ← |π[p_i]|
19   S_i^out ← ∅
20 else
21   lbound_i^out ← 0
22   if M ≠ N then //the input must be isolated singletons
23     i' ← min({j | p_j ∈ C})
24     S_i^out ← {p_i'}
25   else
26     repeat forever
27       S_i^out ← S_i^in
```

Figure 7: Transform $\Pi\Upsilon$ into $\Pi\Omega\Upsilon_{n-1}$

Otherwise, C is the only component that contains correct processes. Because every processes in C set its *lbound*^{out} to 0, ($\Pi\Omega 2$) holds. For ($\Pi\Upsilon 1$), we need to consider the partition scheme of $\Pi\Upsilon$ output. Since every process announces its *cid*^{in} in M before announcing its *cid*^{out} in N, N does not contain $\perp$ cells implies M is also fully filled. So every process is also able to know the partition of the $\Pi\Upsilon$ output. There are two cases: (i) there are at most n components in the $\Pi\Upsilon$ partition; (ii) there are $n+1$ components in the $\Pi\Upsilon$ partition.

For case (i), according to the property *Convergence* of the $converge()$ instance, every process commits to a value at line 5 (no processes crash here because everyone fills its cell in N). According to line 11 of the $converge()$ routine in Figure 8, we know that processes always commit to their own input values. This means the $\Pi\Omega\Upsilon_{n-1}$ partition follows the $\Pi\Upsilon$ partition, making $M = N$. So every process in C copies S^{in} to S^{out} (line 27). Since C is the only component containing correct processes, ($\Pi\Upsilon 2$) of $\Pi\Upsilon$ must hold, so S^{in} on every process in C also satisfies the requirement of ($\Pi\Upsilon 1$) of $\Pi\Omega\Upsilon_{n-1}$.

In case (ii), because of a similar argument in case (i), we know M and N are both fully filled, and $M \neq N$ since there are at most n components in the partition derived from N. Since there are $n+1$

components in the $\Pi\Upsilon$ partition, we know that all of them are singletons. So ($\Pi\Upsilon 2$) cannot hold and ($\Pi C3$) must hold. This means there must be at least two correct processes. Since only C contains correct processes, we know that C contains at least two correct processes. Because every process in C sets its S^{out} to a singleton set that contains the same process (line 24) in C, the singleton set must not be exact set of correct processes. So ($\Pi\Upsilon 1$) holds.

Therefore, in all cases ($\Pi C1$), ($\Pi C2$), ($\Pi\Omega 1$), ($\Pi\Omega 2$), and ($\Pi\Upsilon 1$) holds for the processes' $\Pi\Omega\Upsilon_{n-1}$ output *cid*^{out}, *lbound*^{out}, and S^{out}. □

6.2 Impossible transformations

Proving the impossible transformations is the critical step to establish the results of this paper. Among all the impossible transformations captured by the non-existent directed paths in Figure 1, several of them are critical ones, meaning that their impossibility implies the rest impossible transformations. This is based on the fact that if we show that $\mathcal{C}_1 \not\succeq \mathcal{C}_2$, then for all $\mathcal{C}_3 \preceq \mathcal{C}_1$ and all $\mathcal{C}_4 \succeq \mathcal{C}_2$, we have $\mathcal{C}_3 \not\succeq \mathcal{C}_4$. Each of the following lemmata in this section shows one critical impossible transformation, and together they imply all the impossible transformations known so far.

Many proofs of these lemmata are technically involved, because Υ is already very weak, and thus showing that so many other failure detectors are still incomparable to or strictly weaker than Υ is delicate. For these proofs, it is sometimes convenient to view it as an adversary trying to defeat any possible transformations. The adversary can see the current output generated by a transformation, and it can manipulate the outputs of the failure detector to be transformed and it can crash processes if needed to prevent the transformation from succeeding. In this section, we provide proof outlines to all lemmata using the language of adversary designing strategy to beat the transformation algorithms. In the appendix, we include all technical proofs that match our descriptions in the proof outlines.

Lemma 6 $\Pi\Omega_2$ *cannot be transformed into* $\Pi\Upsilon$*, i.e.,* $\Pi\Omega_2 \not\succeq \Pi\Upsilon$.

Proof Outline. We know that Ω_n can be transformed to Υ easily by taking the complement of the Ω_n output. The reason that this transformation cannot be adapted to $\Pi\Omega_k$ is that $\Pi\Omega_k$ allows a live component P_j in which all processes are eventual leaders and *lbound* stabilizes to $|P_j|$. If we take the complement of the leader set in P_j with respect to P_j we get an empty set. The proof explores this basic idea.

In the case of $\Pi\Omega_2$, suppose for a contradiction that there is a transformation T from $\Pi\Omega_2$ to $\Pi\Upsilon$. The adversary constructs a run in which the $\Pi\Omega_2$ has a partition $\pi = \{P_1, P_2\}$, where $P_1 = \{p\}$. It sets *lbound* of every process to 1 and p's *isLeader* always to *True*, making P_1 a live component of $\Pi\Omega_2$. It will manipulate the *isLeader* outputs for processes in P_2 to create a contradiction. It then run T to see how it partitions the processes for $\Pi\Upsilon$. Let Q_1 be the component containing p with respect to $\Pi\Upsilon$. Once the adversary knows the partition, it crashes all processes not in Q_1. Since only one component left for $\Pi\Upsilon$, ($\Pi\Upsilon 2$) has to be true. This implies that Q_1 contains at least two processes. From now on, whenever the S output of $\Pi\Upsilon$ in Q_1 stabilizes to some subset S_i, the adversary suppresses all processes in $Q_1 \setminus S_i$ (i.e., prohibit these processes from taking any steps) for long enough time to force T to stabilize the S output to a different set $S_{i+1} \neq S_i$, because S_i appears to be the exact set of correct processes. Once T changes the S output, the adversary releases the suppressed processes so that they take some steps, and then it repeats the procedure for S_{i+1}, and so on. The adversary can keep doing so because $Q_1 \setminus S_i$ contains either p or some process in P_2, and thus it can always set *isLeader* of some process in $Q_1 \setminus S_i$ to *True* without violating the $\Pi\Omega_2$ requirement. The result is that the adversary forces T into an infinite run in which only one component Q_1 for $\Pi\Upsilon$ contains correct processes but its S output never stabilizes, a contradiction. □

Lemma 6 implies that for all $\Pi\Omega_k$ with $k \geq 2$, $\Pi\Omega_k$ cannot be transformed into Υ. This is the first key result. Moreover, because $\Pi\Omega_k$ can be transformed into $\Pi\Omega\Upsilon_k$, Lemma 6 further implies that $\Pi\Omega\Upsilon_k$ is strictly weaker than Υ, the second key result of the paper. Next lemma shows another key result of the paper.

Lemma 7 *(1)* $\Pi\Upsilon$ *can be transformed into* Υ *when* $n \leq 2$. *(2)* $\Pi\Upsilon$ *cannot be transformed into* Υ *when* $n \geq 3$.

Proof Outline. For (2), suppose there is a transformation T. The adversary sets up $\Pi\Upsilon$ with two components each with at least two processes and all processes are correct. This satisfies ($\Pi C3$), and thus the adversary is free to manipulate the S output of $\Pi\Upsilon$ at its will. It then uses the technique similar as in the proof of Lemma 6 to construct a run in which the output of Υ never stabilizes. □

Lemma 8 Υ *cannot be transformed into* $\Pi\Omega_n$ *when* $n \geq 2$.

Proof Outline. Suppose there is a transformation T. If the partition of $\Pi\Omega_n$ generated by transformation T contains only a single component, then the proof is the same as proving Υ cannot be transformed into Ω_n in [8]. If the partition of $\Pi\Omega_n$ has at least two components, let P_1 be one of the components. The adversary first sets the Υ output to $P \setminus P_1$, and apply a technique used in [6] to repeatedly suppress the leader processes in all components that are potentially live components for $\Pi\Omega_n$ (these are called *quasi-live components* in the proofs), the purpose of which is to construct an infinite run in which there is no live component. The only way the transformation can counter this measure is by setting the *lbound* outputs of processes in P_1 to $|P_1|$. But the adversary can counter this again by crashing all processes in P_1, setting Υ output to P_1, and re-apply the suppression technique. The result is a run in which no live component exists. The key is that the adversary need to wait until the *lbound* output on P_1 is at least the size of a component to crash the component. This guarantees that the transformation cannot set *lbound* on $P \setminus P_1$ to $|P \setminus P_1|$ to defeat the adversary. □

Lemma 6 and 8 establish that Υ and $\Pi\Omega_k$ with $k \geq 2$ are not comparable. Together with the possible transformations of Lemma 3, they immediately imply that $\Pi\Omega\Upsilon_k$ is strictly weaker than both Υ and $\Pi\Omega_k$ for any $k \geq 2$.

Next lemma establishes that the strength of $\Pi\Omega_k$ (as well as Ω_k) decreases as k increases.

Lemma 9 Ω_k *cannot be transformed into* $\Pi\Omega_{k-1}$ *with* $k \geq 2$.

Proof Outline. Suppose there is a transformation T. The adversary selects k processes to set their *isLeader* to *True*, and set *lbound* on all processes to k. Let Q be the set of k processes whose *isLeader* is set to *True* by the adversary. It then let T run to see how T partitions the processes. Suppose $\{P_1, P_2, \ldots, P_s\}$ is the partition. Next, the adversary go through P_1, $P_2, \ldots$, one by one to do the following. At each component P_j, if at any time it finds that $|P_j \cap Q|$ is at most the *lbound* output generated by T at some process in P_j, then the adversary crashes P_j and goes to P_{j+1}. The adversary will not crash all components in this manner because, by ($\Pi\Omega 1$) of $\Pi\Omega_{k-1}$, the sum of maximum *lbound* outputs of all components is at most $k-1$ while the size of Q is k. So the adversary will stop at some component P_j such that the maximum *lbound* generated by T for processes in P_j is less than $|P_j \cap Q|$. While at this component, the adversary suppresses all other processes not in P_j, and forces T to stabilize to a set of leaders, the number of which is at most the maximum *lbound* value of P_j. Then the adversary can suppress all these leaders and continue the run. It can do so because the number of leader suppressed is less than $|P_j \cap Q|$, and thus some process in $P_j \cap Q$ is not suppressed and there is at least a leader for Ω_k. The adversary repeats such suppression of leader processes generated by T, and between two suppression period, it releases all processes in P_j to make sure they all take steps. Therefore, the adversary forces T into a run in which P_j is the only component containing correct processes, but its leaders never stabilizes, contradicting to the requirement of $\Pi\Omega_{k-1}$. $\square$

Lemma 10 $\Pi\Omega\Upsilon_k$ *cannot be transformed into* $\Pi\Omega\Upsilon_{k-1}$ *for any* $k \geq 1$ *and* $n \geq 2$.

Proof Outline. Suppose there is a transformation T. Consider the case when $k < n$ first. The adversary creates a partition of two components $\{P_1, P_2\}$ for $\Pi\Omega\Upsilon_k$, where P_1 contain k processes with *lbound* $= k$ and $S = \emptyset$ and is the live component in the infinite run. It then uses the similar strategy as in Lemma 9 to defeat the transformation algorithm T. First, it lets T run to see how T partitions the processes. Suppose the partition is $\{P'_1, P'_2, \ldots, P'_s\}$. It then goes through $P'_1, P'_2, \ldots$ one by one. For P'_i, whenever it sees that the *lbound* outputs generated by T in P'_i increases to at least $|P_1 \cap P'_i|$, it crashes P'_i and goes to the next component. Eventually it stops at a component P'_j in which the *lbound* outputs of processes in P'_j are always less than $|P_1 \cap P'_j|$. This is guaranteed by ($\Pi\Omega$1) of $\Pi\Omega\Upsilon_{k-1}$. Then for P'_j, it suppresses all processes not in P'_j to force T to generate $S_i \subseteq P'_j$ and $S_i \neq \emptyset$ and $S_i \neq$ *correct*(P'_j). Whenever this happens, it suppresses processes in $P'_j \setminus S_i$ to force T to generate $S_{i+1} \neq S_i$. It then releases all processes in P'_j to make sure they take steps so that eventually they are correct processes. The adversary can keep doing so because it can manipulate the S output of $\Pi\Omega\Upsilon_k$ for processes in P_2 to make P_2 temporarily look like a live component during any one suppression period. Therefore, eventually the adversary forces a run of T in which only one component P'_j contains correct processes but S outputs never stabilize, a contradiction.

Now consider the case when $k = n$. In this case, the caveat of the above strategy is that P_2 contains only one process and thus the adversary cannot possibly generate a correct S output in P_2 during any suppression period. For this case, the adversary needs to adapt its strategy such that it waits to see how T partitions the processes and then decides how to set *lbound* and S outputs for $\Pi\Omega\Upsilon_n$. It still partitions the processes into P_1 and P_2 with P_1 containing n processes. It initially sets *lbound* of every process to 0, and sets the S output of processes in P_1 to $\{p\}$ with $p \in P_1$. Since P_1 contains at least two processes, this S output makes P_1 a live component. The adversary then lets T run until T outputs all *cid*'s and fixes the partition of $\Pi\Omega\Upsilon_{n-1}$. Let the partition be $\{P'_1, P'_2, \ldots, P'_s\}$, with $P'_s \supseteq P_2$. If $P'_s = P_2$, the adversary sets *lbound* and S outputs of processes in P_1 to n and $\emptyset$, respectively, and sets *lbound* and S outputs of the only process in P_2 to 0 and $\emptyset$; otherwise, it sets *lbound* and S outputs of processes in P_1 to $n-1$ and $P'_s \cap P_1$, respectively, and sets *lbound* and S outputs of the only process in P_2 to 1 and $\emptyset$.

The adversary then uses the same strategy as in the above case of $k < n$ to go through $P'_1, P'_2, \ldots$. If it stops at a component P'_j before P'_s, the adversary already forces a run in which only P'_j contains correct processes but the S outputs of $\Pi\Omega\Upsilon_{n-1}$ never stabilize. Suppose the adversary crashes all other components and only P'_s left. In this case, P'_s cannot be the same as P_2, since otherwise, the sum of *lbound*'s of all previous components must be n, violating ($\Pi\Omega$1) of $\Pi\Omega\Upsilon_{n-1}$. Thus $P'_s \setminus P_2 = P'_s \cap P_1$ is not empty. Note that only one component P'_s is left for $\Pi\Omega\Upsilon_{n-1}$ but it crosses two components P_1 and P_2 for $\Pi\Omega\Upsilon_n$. Moreover, P_2 is a live component with respect to $\Pi\Omega\Upsilon_n$. The adversary can now manipulate the S outputs of processes in $P'_s \setminus P_2$ such that during each suppression period, either P_1 or P_2 looks like a live component for $\Pi\Omega\Upsilon_n$ to force T to change the S output of $\Pi\Omega\Upsilon_{n-1}$. By repeating the suppression period while releasing processes in P'_s between two suppression period, the adversary forces a run in which the S outputs in P'_s never stabilize. $\square$

Lemma 10 together with $\Pi\Omega\Upsilon_k \preceq \Pi\Omega\Upsilon_{k-1}$ implies that $\{\Pi\Omega\Upsilon_0, \Pi\Omega\Upsilon_1, \ldots, \Pi\Omega\Upsilon_n\}$ forms a strictly weakening hierarchy.

Lemma 11 *$\Pi\Omega_{k+1}$ cannot be transformed into $\Pi\Omega\Upsilon_{k-1}$ for any $k \geq 2$.*

Proof Outline. Suppose there is a transformation T. The adversary uses the same approach as in the proof of Lemma 10 for the case of $k < n$. The only difference is that for component P_2, the adversary can always set the *isLeader* of one process in P_2 to *True* because the sum of maximum *lbound* values of each component is $k + 1$ in $\Pi\Omega_{k+1}$. This allows the adversary to make P_2 appear to be a live component temporarily in each suppression period. The full proof is omitted since it is mostly a repetition of the proof of Lemma 10 for the case of $k < n$. □

Lemma 12 *For any $n \geq 3$, (1) $\Pi\Upsilon$ cannot be transformed into $\Pi\Omega\Upsilon_{n-3}$, and (2) $\Pi\Upsilon$ cannot be transformed into $\Pi\Omega\Upsilon_{n-2}$ when n is odd.*

Proof Outline. Suppose there is transformation T. Consider the case of $\Pi\Omega\Upsilon_{n-3}$ first. The adversary partitions the processes such that each component contains exactly two processes, except perhaps one component that contains three processes (when $n + 1$ is odd). It then uses the similar approach as in the proof of Lemma 10. In this case, whenever the adversary sees a component Q of $\Pi\Omega\Upsilon_{n-3}$ such that the maximum *lbound* of Q so far is at least $|Q|$, it crashes the component. By ($\Pi\Omega 1$) of $\Pi\Omega\Upsilon_{n-3}$, at least 4 processes will not be crashed. This implies that there are at least two components of $\Pi\Upsilon$ that contain correct processes in the infinite run, so ($\Pi C3$) are satisfied and the adversary are free to set S outputs at its will. Then the adversary uses the same technique of suppressing processes as in the proof of Lemma 10 to make sure there is no live components in the infinite run.

The same proof can be extended to the case of $\Pi\Omega\Upsilon_{n-2}$ when n is odd, since P contains an even number of processes and all components of $\Pi\Upsilon$ contains exactly two processes. In this case, at least 3 processes will not be crashed by the adversary, so at least two components of $\Pi\Upsilon$ contain correct processes. □

In conclusion, Theorem 1 is implied by Lemma 1(1)(3), Lemma 2, Lemma 6, Lemma 8 and Lemma 9. Theorem 3 is implied by Lemma 1(2)(4), Lemma 4, Lemma 10 and Lemma 11. Theorem 5 is implied by Lemma 1(5), Lemma 5, Lemma 6, Lemma7, Lemma 8 and Lemma 12.

There are still a couple open problems left before we can completely characterize all relationships in Figure 1. They are: (a) whether $\Pi\Omega_k$ can be transformed into $\Pi\Omega\Upsilon_{k-1}$ for any $k \geq 2$; and (b) whether $\Pi\Upsilon$ can be transformed into $\Pi\Omega\Upsilon_{n-2}$ when n is even. We conjecture that all these transformations are impossible. If so, Figure 1 is indeed a full characterization of all relationships.

7 Concluding Remarks

All of our partitioned failure detectors use static partitions, which means the partition cannot be changed once the *cid* outputs are fixed to non-$\perp$ values. In [6] we also propose dynamically splittable partitioned failure detectors Π_k^S that further weakens statically partitioned failure detectors Π_k in the message-passing model. However, it is not clear how to adapt this approach to weaken the statically partitioned failure detectors defined in this paper. This is left as future research.

The discovery of failure detectors even weaker than Υ may suggest that the conjecture made in [8] that n-set agreement is the minimum decision task in terms of minimum information required is not true. This is another research direction to see if there is any other decision task strictly weaker than n-set agreement in terms of failure information needed to solve the problem.

References

[1] E. Borowsky 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. ACM Press, May 1993.

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

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

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

[5] W. Chen, J. Zhang, Y. Chen, and X. Liu. Failure detectors and extended Paxos for k-set agreement. Technical Report MSR-TR-2007-48, Microsoft Research, May 2007.

[6] W. Chen, J. Zhang, Y. Chen, and X. Liu. Partition approach to failure detectors for k-set agreement. Technical Report MSR-TR-2007-49, Microsoft Research, May 2007.

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

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

[9] M. Herlihy and L. D. Penso. Tight bounds for k-set agreement with limited scope accuracy failure detectors. *Distributed Computing*, 18(2):157–166, 2005.

[10] M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, 1999.

[11] M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Prog. Lang. Syst.*, 12(3):463–492, July 1990.

[12] P. Jayanti. Robust wait-free hierarchies. *J. ACM*, 44(4):592–614, 1997.

[13] A. Mostefaoui, 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.

[14] A. Mostefaoui, 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*, pages 153–162, July 2006.

[15] A. Mostefaoui 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, July 2000.

[16] M. Raynal and C. Travers. In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. Technical Report TR 06-1811, INRIA, Aug. 2006.

[17] M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM Journal on Computing*, 29(5):1449–1483, 2000.

[18] J. Yang, G. Neiger, and E. Gafni. Structured derivations of consensus algorithms for failure detectors. In *Proceedings of the 17th ACM Symposium on Principles of Distributed Computing*, pages 297–306, June 1998.

Appendix

A The $converge()$ routine

The complete $converge()$ routine is shown in Figure 8. It is exactly the same as the original one in [18], except that the interface is changed.

function $converge(\ell, p, v)$
Shared variables $a[1 \ldots n+1]$ and $b[1 \ldots n+1]$, initially $\perp$

1 $a[p] \leftarrow v$
2 **for** $q = 1$ **to** $n+1$
3 $\quad r[q] \leftarrow a[q]$
4 **if** $|\{r[q] | r[q] \neq \perp\}| \leq \ell$ **then**
5 $\quad b[p] \leftarrow$ *True*
6 **else**
7 $\quad b[p] \leftarrow$ *False*
8 **for** $q = 1$ to $n+1$
9 $\quad s[q] \leftarrow b[q]$
10 **if** $\forall q(s[q] \neq \textit{False})$ **then**
11 $\quad$ **return** $(\textit{True}, v)$
12 **else if** $\exists q(s[q] = \textit{True})$ **then**
13 $\quad$ **return** $(\textit{False}, a[q])$ such that $s[q] = \textit{True}$
14 **else**
15 $\quad$ **return** $(\textit{False}, v)$

Figure 8: The extended converge algorithm allowing processes to use different converge number ℓ.

Theorem 7 *The algorithm in Figure 8 is a correct implementation of* $converge()$.

Proof. It is straightforward that *C-Termination* and *C-Validity* hold. The *Convergence* property is also easy to show, since every process p will set $b[p]$ to *True*.

We now prove the *C-Agreement* property. Let ℓ_{max} be the maximum ℓ value that processes pass into the $converge()$ instance. We say that a value v is *proposed* if some process p writes v into the array $a[p]$ and writes *True* into $b[p]$. It is easy to see that only the first ℓ_{max} values written to the array $a[\,]$ could be proposed, and processes only commits to the proposed values. Thus, to show *C-Agreement*, we only need to check the case when p commits to a value v while q picks a value v' but q does not commit to v'. Since p commits, $b[p]$ must be *True* and p does not find any *False* flags in array $b[\,]$. Since q does not commit, it must perceive at least one *False* flag in $b[\,]$. This implies q must see $b[p] = \textit{True}$. Therefore, q picks a proposed value at line 13. Since all picked values are proposed values and there are at most ℓ_{max} proposed values, *C-Agreement* holds. □

B Impossible transformations in Section 6

Lemma 6 $\Pi\Omega_2$ *cannot be transformed into* $\Pi\Upsilon$*, i.e.,* $\Pi\Omega_2 \not\succeq \Pi\Upsilon$.

Proof. Suppose, for a contradiction, that we have an algorithm T that transforms any failure detector $\mathcal{D}$ in $\Pi\Omega_2$ to a failure detector $\mathcal{D}'$ in $\Pi\Upsilon$. Let $(\textit{isLeader}, \textit{lbound}, \textit{cid})$ denote the output of $\mathcal{D}$, and $(S, \textit{cid}')$ denote the output of $\mathcal{D}'$ generated by algorithm T.

We consider a partition of P, $\pi = \{P_1, P_2\}$, such that $P_1 = \{p\}$. We consider a failure pattern in which p is a correct process. The failure pattern for other processes will be determined shortly. We construct a failure detector history H of $\mathcal{D}$ such that (a) *lbound* outputs of all processes are always 1; (b) for process p, p's *isLeader* outputs are always *True*, p's *cid* outputs are always 1; and (c) for process q other than p, q's *cid* outputs are always 2. And, at any time, exactly one process in P_2 outputs *isLeader* $=$ *True*. We can see that in H, component P_1 must be a live component, so it is an admissible failure detector history of $\mathcal{D}$. Next we will manipulate the *isLeader* outputs of processes in P_2 to reach a contradiction.

To achieve the objective, we construct run R as follows. Initially the *isLeader* outputs of processes in P_2 could be arbitrary, and we let T run to some time t_0 at which the outputs (S, cid') of $\mathcal{D}'$ are generated by T and $\mathit{cid}' \neq \perp$. By $\Pi C1$, all correct processes eventually have $\mathit{cid}' \neq \perp$, so such time t_0 exists. From the cid' outputs, it is clear how processes are partitioned with respect to $\mathcal{D}'$. Let Q_1 be the component containing process p. We then crash all processes not in Q_1 in run R at time $t_0 + 1$. Thus the failure pattern for run R is such that all processes in Q_1 are correct and all processes not in Q_1 crash at time $t_0 + 1$. Since there is only one component containing correct processes, ($\Pi C3$) does not hold. Therefor ($\Pi\Upsilon 2$) must hold, which implies immediately that Q_1 contains at least two processes. The rest of the proof is essentially the proof that $\Pi\Omega_2$ cannot be transformed into Υ.

We let T continue to run after the crash of processes not in Q_1. By property ($\Pi\Upsilon 2$), eventually all processes in Q_1 output the same nonempty set $S \subseteq Q_1$, and S is not the set of correct processes in Q_1. Let t_1 be such a time and S_1 be the output. After time t_1, we suppress all processes in $Q_1 \setminus S_1$ (i.e., prohibit these processes from taking any steps). There are two cases. In the first case, $p \in S_1$, which means p is not suppressed. Then component P_1 is still a live component with respect to $\mathcal{D}$ and thus we can set *isLeader* outputs of other processes arbitrarily and continue the run of T. In the second case, $p \notin S_1$, i.e., p is suppressed. In this case, $S_1 \subseteq P_2$, and we select one process $q \in S_1$ and set its *isLeader* to *True* and set the *isLeader* of all other processes in P_2 to *False*. Thus P_2 is a live component with respect to $\mathcal{D}$. We then continue the run of T. In either case, processes in S_1 cannot distinguish this run from a run in which all processes not in S_1 indeed have crashed. So by ($\Pi\Upsilon 2$) eventually at some time t_2 processes in S_1 output a nonempty set $S_2 \subseteq Q_1$ such that S_2 is not the set of correct processes in Q_1, i.e. $S_2 \neq S_1$. After t_2, we let all processes in $Q_1 \setminus S_1$, those previously suppressed, to take at least one step each. We then suppress all processes in $Q_1 \setminus S_2$ and repeat the above process.

We can repeat the above procedure infinitely many times. In the resulting run R, (a) all processes in Q_1 take infinitely many steps so they are all correct; (b) component P_1 is a live component with respect to $\mathcal{D}$; and (c) there are infinite time points $t_1, t_2, \ldots$ and infinite number of sets $S_1, S_2, \ldots$ such that some process outputs S_i at time t_i for failure detector $\mathcal{D}'$ and $S_{i+1} \neq S_i$, for all $i \geq 1$. Therefore in run R, the S output of $\mathcal{D}'$ is not stable in component Q_1. This violates property ($\Pi\Upsilon 2$) since Q_1 is the only component containing correct processes. □

Lemma 7 *(1)* $\Pi\Upsilon$ *can be transformed into* Υ *when* $n \leq 2$. *(2)* $\Pi\Upsilon$ *cannot be transformed into* Υ *when* $n \geq 3$.

Proof. For part (1), we present a transformation algorithm from a failure detector $\mathcal{D}$ in $\Pi\Upsilon$ into a failure detector $\mathcal{D}'$ in Υ. Suppose (S, cid) is the output of failure detector $\mathcal{D}$ in $\Pi\Upsilon$ and S' is the output of failure detector $\mathcal{D}$ in Υ. For process p_i, there are three cases. Case 1: If it finds there exists some process with $\mathit{cid} = \perp$, set $S' = \{p_i \mid p_i.\mathit{cid} = \perp\}$. Case 2: If there is only one component with $n+1$ processes in $\Pi\Upsilon$, p_i set $S' = S$. Case 3: If there are at least two components in $\Pi\Upsilon$, since $n + 1 \leq 3$, at least one component contains only one process. Then p_i set S' be such singleton component. If there are several singleton components, use process ID to break the tie. It is obviously that all correct processes have the same and stable output S'. In case 1 and 2, it is easy to prove S' is not the set of exact correct processes. In case 3, if S' is the set of exact correct processes, this means only a singleton component does not crash. But in such component, $\mathcal{D}$ cannot give a correct output of S. Thus, the transformation algorithm is correct.

For part (2), suppose there exists a transformation algorithm T from a failure detector $\mathcal{D}$ in $\Pi\Upsilon$ into a failure detector $\mathcal{D}'$ in Υ. Suppose (S, cid) is the output of failure detector $\mathcal{D}$ in $\Pi\Upsilon$ and S' is failure detector $\mathcal{D}$ in Υ. We consider a partition $\{P_1, P_2\}$ of P with $|P_1| \geq 2$ and $|P_2| \geq 2$ which is possible when $n \geq 3$.

Then, we construct a run R in which all processes are correct, so by ($\Pi C3$) that S output of all processes could be arbitrary. By the specification of Υ, eventually all processes output the same nonempty set S', and S' is not the set of correct processes. Let t_1 be such a time and S_1 be the output. After time t_1, we suppress all processes in $P \setminus S_1$ (i.e., prohibit these processes from taking any steps). And set $S \neq S_1 \cap P_1$ for processes in P_1 and set $S \neq S_1 \cap P_2$ for processes in P_2. Since $|P_1| \geq 2$ and $|P_2| \geq 2$, it is always possible to find suitable non-empty S output of all processes. We then continue the run of T. Processes in S_1 cannot distinguish this run from a run in which all processes not in S_1 indeed have crashed. So by the specification of Υ, eventually at some time t_2 processes in S_1 output a nonempty set $S_2 \subseteq P$ such that S_2 is not the set of correct processes, i.e. $S_2 \neq S_1$. After t_2, we let all processes take at least one step each. We then suppress all processes in $P \setminus S_2$ and repeat the above process.

We can repeat the above procedure infinitely many times. In the resulting run R, (a) all processes take infinitely many steps, so they are all correct; (b) there are infinite time points $t_1, t_2, \ldots$ and infinite number of sets $S_1, S_2, \ldots$ such that some process outputs S_i at time t_i for failure detector $\mathcal{D}'$ and $S_{i+1} \neq S_i$, for all $i \geq 1$. Therefore in run R, the S' output of $\mathcal{D}'$ is not stable. This violates the specification of Υ. □

Lemma 8 *Υ cannot be transformed into $\Pi\Omega_n$ when $n \geq 2$.*

Proof. Suppose, for a contradiction, that we have an algorithm T that transforms any failure detector $\mathcal{D}$ in Υ to a failure detector $\mathcal{D}'$ in $\Pi\Omega_n$. Let S denote the output of $\mathcal{D}$, and $(\mathit{isLeader}, \mathit{lbound}, \mathit{cid})$ denote the output of $\mathcal{D}'$ generated by algorithm T.

Firstly, set $S = \{p\}$ for arbitrary process p and we let T run to some time t_0 at which the outputs $(\mathit{isLeader}, \mathit{lbound}, \mathit{cid})$ of $\mathcal{D}'$ are generated by T and $\mathit{cid} \neq \perp$. By $\Pi C1$, all correct processes eventually have $\mathit{cid} \neq \perp$, so such time t_0 exists. From the *cid* outputs, it is clear how processes are partitioned with respect to $\mathcal{D}'$. If there is only one component, that is, no partition occurs, we can apply the same proof as that of Theorem 1 in [8] to reach a contradiction, since with only one component $\Pi\Omega_n$ collapses into Ω_n. So we only consider the case in which the output of $\mathcal{D}'$ has at least two components. Suppose the partition is $\{P_1, P_2, \cdots, P_s\}$ with $s \geq 2$.

Let F be a failure pattern, H' be the failure detector history of $\mathcal{D}'$ generated by T under failure pattern F. We define $P_i.\mathit{lbound}(t) = \max\{H'(p, t').\mathit{lbound} \,|\, t' \leq t, p \in P_i \setminus F(t')\}$. We define $A(P_i, t) = \{p \in P_i \setminus F(t) \,|\, H'(p, t).\mathit{isLeader}' = \mathit{True}\}$. We say that component P_i is *quasi-live* at time t if $|A(P_i, t)| \leq P_i.\mathit{lbound}(t)$. Note that for a live component P_i, there exists a time after which P_i is always quasi-live. We define $A(t)$ to be the union of $A(P_i, t)$'s where P_i is a quasi-live component. Now, we construct two possible infinite sequences of runs by the following inductive process.

Possibility 1. Set $S = P \setminus P_1$.

Run R_0: no process crashes in this run. Define t_0 such that $A(t_0) \neq \emptyset$.

Run R_1: R_1 runs exactly the same as in R_0 until time t_0. Because we do not crash any processes yet, $F(t) = \emptyset$ for all $t \leq t_0$. Let $A(t_0)$ be as defined above in run R_1. If $P_1.\mathit{lbound}(t_0) \geq |P_1|$, go to Possibility 2. Otherwise, at time $t_0 + 1$, we crash all processes in $A(t_0)$. So for all $t \geq t_0 + 1$, $F(t) \equiv A(t_0)$. We then continue the run of algorithm T to find a time $t_1 > t_0 + 1$, by which every correct process has taken at least one step after time $t_0 + 1$. If P_1 is not a quasi-live component at t_0, then $P_1 \cap A(t_0) = \emptyset$. If P_1 is a quasi-live component at t_0, then $|A(P_1, t_0)| \leq P_1.\mathit{lbound}(t_0)$, and since $P_1.\mathit{lbound}(t_0) < |P_1|$, we know that $P_1 \setminus A(t_0) \neq \emptyset$. Thus, in either case, at least one process in P_1 does not crash in run R_1. Therefore, $S = P \setminus P_1$ is an admissible output of Υ in run R_1.

In general, we try to construct R_i based on R_{i-1} for all $i \geq 2$. In R_{i-1}, there are two critical time points t_{i-2} and t_{i-1}. The failure pattern in R_{i-1} is $F(t) = \emptyset, \forall t \leq t_{i-2}$ and $F(t) = A(t_{i-2}), \forall t \geq t_{i-2} + 1$. Every process not in $A(t_{i-2})$ has taken at least one step between $t_{i-2} + 1$ and t_{i-1}. R_i is constructed as the following.

Run R_i: R_i runs exactly the same as in R_{i-1} until time t_{i-2}. From $t_{i-2} + 1$ to t_{i-1}, instead of crashing the processes in $A(t_{i-2})$, we hold these processes and do not let them take any steps in R_i. All the other processes simulate their execution as in R_{i-1} until t_{i-1}. Now we have a simulated "R_{i-1}" at the beginning of R_i, with a different failure pattern: $F(t) = \emptyset, \forall t \leq t_{i-1}$. Since the algorithm is deterministic, at time t_{i-1} process and shared object states are exactly the same as in run R_{i-1}.

During the execution between $t_{i-2}+1$ and t_{i-1}, we calculate $A(t_{i-1})$ in a similar manner as described in R_1. If $P_1.\mathit{lbound}(t_{i-1}) \geq |P_1|$, go to Possibility 2. Otherwise, we crash the processes in $A(t_{i-1})$ at $t_{i-1}+1$, and let the processes not crashed run. So the failure pattern after $t_{i-1} + 1$ is $F(t) = A(t_{i-1}), \forall t \geq t_{i-1} + 1$. Let t_i be the time by which every correct process in R_i has taken at least one step after $t_{i-1} + 1$. Since $P_1.\mathit{lbound}(t_{i-1}) < |P_1|$, so at least one process in P_1 does not crash. Therefore, $S = P \setminus P_1$ is an admissible output of Υ in run R_i.

If $P_1.\mathit{lbound}(t_i) < |P_1|$ for all $i \geq 0$, then we have constructed an infinitely series of runs $R_0, R_1, R_2, \ldots$. Let $R_\infty = \lim_{i\to\infty} R_i$. That is, for any i, let the failure detector history and the sequence of steps of run R_∞ be identical to the run R_i until time t_{i-1}. We need to show that R_∞ is still a legitimate run of algorithm T with some failure detector.

We start by defining the failure pattern F of R_∞ in the following way. For every process p, there are two possible cases. In the first case, there exists j such that for all $i \geq j$, p crashes in run R_i. Let j_p be the smallest such value. Then we define that in run R_∞, p crashes at time $t_{j_p-1} + 1$. For all processes that do not belong to the first case, they are correct in run R_∞.

Now we show R_∞ is a legitimate run of algorithm T under the failure pattern F. First, we need to show that the failure pattern F derived above does not make the output S of $\mathcal{D}$ violate the property of Υ. Since $P_1.\mathit{lbound}(t_i) < |P_1|$ for all i, there exists at least one process $p \in P_1$ correct in run R_i. Therefore, at least one process $p \in P_1$ is a correct process in run R_∞. Then, $S = P \setminus P_1$ is not the exact set of correct processes in run R_∞.

Second, we need to verify that in run R_∞, all correct processes take an infinite number of steps. Suppose p is a correct process in R_∞. By its definition, for any time t, there is a $j \geq 1$ such that $t_{j-1} > t$ and p is a correct process in run R_j. By the construction of R_j, we know that p must take at least one step after t_{j-1} and by time t_j. Then we know that p must take at least a step in run R_∞ after t_{j-1} and by time t_j. This implies immediately that p takes an infinite number of steps in R_∞.

Therefore, by the above arguments, we know that R_∞ is a legitimate run of algorithm T with a failure detector $\mathcal{D}$ in Υ. Then we know that T should generate correct outputs of $\mathcal{D}'$ in $\Pi\Omega_n$. This means that eventually there is a live component P_j w.r.t. $\mathcal{D}'$ and correct process $p \in P_j$ such that there is a time t after which P_j is always quasi-live and *isLeader*$'$ of p is always *True*. Thus, for all runs R_l such that $t_{l-1} > t$ and $l \geq u$, we know that P_j is a quasi-live component in R_l, and *isLeader*$'$ of p at t_{l-1} is *True* in R_l. Since $l \geq u$, $A(t_{l-1})$ can be calculated and $p \in A(t_{j-1})$. So p will be crashed in R_l. By our definition of F, p is crashed in R_∞ at some time. Therefore, we reach a contradiction.

Possibility 2: If for some $i \geq 1$, we find $P_1.\mathit{lbound}(t_{i-1}) \geq |P_1|$ in run R_i, we construct another infinitely sequence runs $R'_0, R'_1, \cdots$ as follows.

Run R'_0: R'_0 runs exactly the same as in R_i until time t_{i-1}. At time $t_{i-1}+1$, crash all processes in P_1, set $S = P_1$ and let the processes not crash run. In run R'_0, the failure patter is $F(t) = \emptyset, \forall t \leq t_{i-1}$ and $F(t) = P_1, \forall t \geq t_{i-1}+1$. Since S contains faulty process, R_0 is a legitimate run of algorithm T with a failure detector $\mathcal{D}$ in Υ. Then, we find $t'_0 > t_{i-1}$ such that $A(t'_0) \neq \emptyset$.

Then, we use the same inductive process as for Possibility 1 to construct run $R'_1, R'_2, \cdots$ with the output S of Υ be $P \setminus P_1, \forall t \leq t_{i-1}$ and $P_1, \forall t > t_{i-1}$. Note that by the property ($\Pi\Omega 1$) of $\Pi\Omega_n$, we have $\sum_{j=1}^{s} \textit{lbound}(P_j) \leq n$. Since $P_1.\textit{lbound}(t_{i-1}) \geq |P_1|$, there must exist at least one other component P_j ($j \geq 2$) such that $P_j.\textit{lbound}(t) \leq \textit{lbound}(P_j) < |P_j|$ for all $t \geq 0$. Thus, it is easy to see for any run R'_i at least one process in P_j is correct in the run. We then use the same method to construct a legitimate run R'_∞ based on infinitely sequence of runs $R'_0, R'_1, \cdots$. We can also prove that R'_∞ is a legitimate run of algorithm T under a failure pattern F' defined in the same way as F, in which at least one process is correct. However, by the same argument as for run R_∞, we can argue that there is no live component in run R'_∞. This violates the liveness property of $\Pi\Omega_n$. □

Lemma 9 *Ω_k cannot be transformed into $\Pi\Omega_{k-1}$ with $k \geq 2$.*

Proof. Suppose, for a contradiction, there exists a transformation algorithm T that transforms any failure detector $\mathcal{D}$ in Ω_k to a failure detector $\mathcal{D}'$ in $\Pi\Omega_{k-1}$. The output of $\mathcal{D}$ is denoted by (*lbound*, *isLeader*), and the output of $\mathcal{D}'$ is denoted by (*cid*$'$, *lbound*$'$, *isLeader*$'$). Assume the set of processes is $P = \{p_1, p_2, \ldots, p_{n+1}\}$. Let $Q = \{p_1, p_2, \ldots, p_k\}$. Let F be a failure pattern. H be an output history of $\mathcal{D}$, and H' be an output history of $\mathcal{D}'$.

In the following, we are going to construct a sequence of runs. In each run, we let H be as (1) $H(p_i, t).\textit{lbound} = k$ for all i and t; (2) $H(p_i, t).\textit{isLeader} = \textit{True}$, for all t and $p_i \in Q$; (3) $H(p_i, t).\textit{isLeader} = \textit{False}$, for all t and $p_i \notin Q$. Informally, Q is the set of leaders in $\mathcal{D}$, and the output of $\mathcal{D}$ is stable at the beginning.

We go through the components P_1, $P_2, \ldots$ one by one. In the series of runs $R_{j,m}$ with $m \geq 0$, components $P_1, \ldots, P_{j-1}$ have been crashed because their *lbound* outputs already exceed the number of processes both in Q and in the component. In series $R_{j,m}$, components $P_{j+1}, \ldots, P_s$ are also crashed because we want to isolate component P_{j+1} to run. Each run in this series is an extension of the previous one, and it forces some *isLeader* output in P_j to change. If at some point in this series, the *lbound* output of P_j also exceeds the number of processes in both Q and P_j, then we crash P_j and start the series $R_{j+1,m}$. Eventually, this ends at some series $R_{j,m}$. This final series corresponds a run R^∞ in which only processes in P_j are correct, but the *isLeader* outputs of processes in P_j never stabilizes. This is the contradiction we want. We now give the details of this construction of series of runs.

Run R_0: We let all the processes to run as if all of them are correct, according to ($\Pi C1$) eventually each process needs to output non-$\perp$ values for *cid*$'$. Let t_0 be the time at which $H'(p, t).\textit{cid}' \neq \perp$ for all $p \in P$. Let $\pi = \{P_1, P_2, \ldots, P_s\}$ be the derived partition according to *cid*$'$ at time t_0. According to ($\Pi C2$), We know that *cid*$'$ will not change after t_0. Let $\textit{lbound}'(P_j, t) = \max\{H'(p, t').\textit{lbound}' | t' \leq t, p \in P_j \setminus F(t')\}$. For convenience we assume $\max \emptyset = 0$. Let $\textit{Leaders}'(P_j, t) = \{p | H'(p, t).\textit{isLeader}' = \textit{True}, p \in P_j \setminus F(t')\}$.

Run $R_{1,0}$: $R_{1,0}$ runs exactly the same as R_0 until t_0. Let $t_1 = t_0$. If $\textit{lbound}'(P_1, t_1) \geq |P_1 \cap Q|$, we turn to run $R_{2,0}$. Otherwise, we crash all processes in $P \setminus P_1$ at time $t_1 + 1$ and let processes in P_1 run. Because $\textit{lbound}'(P_1, t_1) < |P_1 \cap Q|$ implies at least one process in Q is in P_1, H is still a legitimate Ω_k output in this run. Then we wait for a time $t_{1,0} > t_1 + 1$ at which P_1 becomes quasi-live. A component P_j is quasi-live at time t if and only if. $\textit{lbound}'(P_j, t) \geq |\textit{Leaders}'(P_j, t)|$ and $|\textit{Leaders}'(P_j, t)| > 0$. Since P_1 is the only component having correct process, $t_{1,0}$ must exist. If $\textit{lbound}'(P_1, t_{1,0}) \geq |P_1 \cap Q|$, we turn to run $R_{2,0}$. Otherwise, we crash all process in *Leaders*$'(P_1, t_{1,0})$ at time $t_{1,0} + 1$. Since $|$*Leaders*$'(P_1, t_{1,0})| \leq$ *lbound*$'(P_1, t_{1,0}) < |P_1 \cap Q|$, at least one process in Q does not crash and H is still legitimate in this run. Then we wait for a time $t'_{1,0} > t_{1,0} + 1$ at which P_1 becomes quasi-live again, and turn to run $R_{1,1}$.

Run $R_{j,m}$ $(1 \leq j \leq s$, $m \geq 1)$: $R_{j,m}$ runs exactly the same as $R_{j,m-1}$ until $t_{j,m-1}$. At time $t_{j,m-1} + 1$, instead of crashing processes in *Leaders*$'(P_{j-1}, t_{j,m-1})$, we suppress the execution of these processes and schedule the other processes to take exactly the same sequence of steps as in $R_{j,m-1}$ until $t'_{j,m-1}$. After $t'_{j,m-1}$, we allow all processes in P_j to run until a time $t_{j,m}$ such that (1) every process in P_j takes at least one step between $t'_{j,m-1}$ and $t_{j,m}$; and (2) P_j becomes a quasi-live component at time $t_{j,m}$. If *lbound*$'(P_j, t_{j,m}) \geq |P_j \cap Q|$, we turn to run $R_{j+1,0}$. Otherwise, *isLeader* output of some processes in P_j must have changed. We then crash all process in *Leaders*$'(P_j, t_{j,m})$ at time $t_{j,m} + 1$. Since $|$*Leaders*$'(P_j, t_{j,m})| \leq$ *lbound*$'(P_j, t_{j,m}) < |P_j \cap Q|$, at least one process in Q does not crash and H is still legitimate in this run. Then we wait for a time $t'_{j,m} > t_{j,m} + 1$ at which P_j becomes quasi-live again, and turn to run $R_{j,m+1}$.

Run $R_{j,0}$ $(2 \leq j \leq s)$: $R_{j,0}$ runs exactly the same as $R_{j-1,0}$ until time t_{j-1}. We know components P_i $(1 \leq i < j-1)$ has been crashed at time t_{j-1} with *lbound*$'(P_i, t_{j-1}) \geq |P_i \cap Q|$. If *lbound*$'(P_{j-1}, t_{j-1}) \geq |P_{j-1} \cap Q|$, let $t_j = t_{j-1} + 1$. Otherwise, it must be true that a run $R_{j-1,m_{j-1}}$ exists such that *lbound*$'(P_j, t_{j-1,m_{j-1}}) \geq |P_{j-1} \cap Q|$. At time $t_{j-1} + 1$ instead of crashing processes in $P_{j'}$ $(j' \geq j)$, we suppress these process from running and allow the processes in P_{j-1} to take exactly the same sequence of steps as in run $R_{j-1,m_{j-1}}$ until time $t_{j-1,m_{j-1}}$. After time $t_{j-1,m_{j-1}}$, we allow every process in $P_{j'}$ $(j' \geq j-1)$ to run, and wait for everyone to take at least one step. Let $t_j > t_{j-1,m_{j-1}}$ be the time such that everyone in $P_{j'}$ takes at least one step by $t_j - 1$. At time t_j we further crash processes in component P_{j-1}. Now we have crash all processes in P_i $(1 \leq i \leq j-1)$. Since we have *lbound*$'(P_i, t_j)) \geq |P_i \cap Q|$ and $\sum_{i=1}^{j-1}$ *lbound*$'(P_i) \leq k-1$ (ensured by $(\Pi\Omega 1)$ for $\Pi\Omega_{k-1}$), $|Q| = k$ implies at least one process in Q is not crashed. So H is still a legitimate output for Ω_k in this run.

Now we look at component P_j. If *lbound*$'(P_j, t_j) \geq |P_j \cap Q|$, we turn to run $R_{j+1,0}$. Otherwise, we crash all processes in $P_{j'}$ $(j < j' \leq s)$ at time $t_j + 1$ and let processes in P_j run. Because *lbound*$'(P_j, t_j) < |P_j \cap Q|$ implies at least one process in Q is in P_j, H is still a legitimate Ω_k output in this run. Then we wait for a time $t_{j,0} > t_j + 1$ at which P_j becomes quasi-live. Since P_j is the only component having correct process, $t_{j,0}$ must exist. If *lbound*$'(P_j, t_{j,0}) \geq |P_j \cap Q|$, we turn to run $R_{j+1,0}$. Otherwise, we crash all process in *Leaders*$'(P_j, t_{j,0})$ at time $t_{j,0} + 1$. Since $|$*Leaders*$'(P_j, t_{j,0})| \leq$ *lbound*$'(P_j, t_{j,0}) < |P_j \cap Q|$, at least one process in Q does not crash and H is still legitimate in this run. We wait for a time $t'_{j,0} > t_{j,0} + 1$ at which P_j becomes quasi-live again, and turn to run $R_{j,1}$.

In the above, we have constructed a series of runs, in which our H is always a legitimate output of Ω_k. We claim that there exist a $j \in [1 \ldots s]$, such that *lbound*$'(P_j, t_j)) < |P_j \cap Q|$ in run $R_{j,0}$ and *lbound*$'(P_j, t_{j,m}) < |P_j \cap Q|$ for all $m \geq 0$ in run $R_{j,m}$. Otherwise, for all j either *lbound*$'(P_j, t_j)) \geq |P_j \cap Q|$ in run $R_{j,0}$, or there exists a m_j such that *lbound*$'(P_j, t_{j,m_j}) \geq |P_j \cap Q|$ in run R_{j,m_j}. Then there must be a run R_{s,m_s}, in which *lbound*$'(P_j, t) \geq |P_j \cap Q|$ for all $j \in [1 \ldots s]$ at some time t. However, $sum_{j=1}^{s} |P_j \cap Q| = k$. This is contradictory to the fact that $sum_{j=1}^{s}$*lbound*$'(P_j, t) \leq k-1$ for all t. So our claim holds. Now let R^∞ be the infinite run such that every $R_{j,m}$ shares a prefix with it until $t_{j,m}$. In this run, all processes in P_j are correct and all processes not in P_j crash eventually, but the *isLeader*$'$ output on processes in P_j cannot stabilize. Therefore, T is not a correct transformation algorithm. □

Lemma 10 $\Pi\Omega\Upsilon_k$ *cannot be transformed into* $\Pi\Omega\Upsilon_{k-1}$ *for any* $k \geq 1$ *and* $n \geq 2$.

Proof. Suppose, for a contradiction, there exists a transformation algorithm T that transforms any failure detector $\mathcal{D}$ in $\Pi\Omega\Upsilon_k$ to a failure detector $\mathcal{D}'$ in $\Pi\Omega\Upsilon_{k-1}$. The output of $\mathcal{D}$ is denoted by (*cid*, *lbound*, S), and the output of $\mathcal{D}'$ is denoted by (*cid*$'$, *lbound*$'$, S'). Assume the set of processes is $P = \{p_1, p_2, \ldots, p_{n+1}\}$. Let $Q = \{p_1, p_2, \ldots, p_k\}$. Let F be a failure pattern. H be an output history of $\mathcal{D}$, and H' be an output history of $\mathcal{D}'$.

We split the proof of this lemma into two parts: 1. $k < n$; 2. $k = n$.

Part 1. In this part, $k < n$, and we let D to output the following initially: (a) The processes are partitioned into two components P_1 and P_2 such that $P_1 = Q$ and $P_2 = P \setminus Q = \{p_{k+1}, \ldots, p_{n+1}\}$. (b) All processes in P_1 set their *lbound* and S to k and $\emptyset$. (c) All processes in P_2 set their *lbound* and S to 0 and $\{p_{n+1}\}$.

Run R_0: We let all process in P to run until T outputs *cid*$'$. This is possible because the initial output of D is valid for $\Pi\Omega\Upsilon_k$. Suppose at time t_0, every process outputs their *cid*$'$, and $\pi' = \{P'_1, P'_2, \ldots, P'_s\}$ is the partition derived from the outputs. To make the proof easier, we assume the components in π' are sorted according to how many processes in Q they contain. Formally, we want $|P'_1 \cap Q| \geq |P'_2 \cap Q| \geq \ldots \geq |P'_s \cap Q|$. In the following, we design a set of runs $R_{j,m}$ for $1 \leq j \leq s$ and $m \geq 0$.

Run $R_{1,0}$: R_1 runs exactly the same as R_0 until time t_0. Let $t_1 = t_0$. If *lbound*$'(P'_1, t_1) \geq |P'_1|$, we turn to run $R_{2,0}$. Otherwise, we crash all processes in $P \setminus P'_1$ at time $t_1 + 1$, and let processes in P'_1 run. Since our sorting ensures $P'_1 \cap Q \neq \emptyset$ ($|Q| = k \geq 2$), the $\mathcal{D}$ output is still legitimate for $\Pi\Omega\Upsilon_k$. Then we wait for a time $t_{1,0} > t_1 + 1$ at which P'_1 becomes quasi-live. Since P'_1 is the only component having correct process, $t_{1,0}$ must exist. Since we have not crashed any process in P'_1 yet, *lbound*$'(P'_1, t_{1,0}) \geq |P'_1 \setminus F(t_{1,0})|$ implies *lbound*$'(P'_1, t_{1,0}) \geq |P'_1|$. If *lbound*$'(P'_1, t_{1,0}) \geq |P'_1|$, we turn to run $R_{2,0}$. Otherwise, $P'_1.S'(t_{1,0})$ must be a non-empty set. We crash all process in $P'_1 \setminus P'_1.S'(t_{1,0})$ at time $t_{1,0} + 1$. If $Q \cap P'_1.S'(t_{1,0}) \neq \emptyset$, the $\mathcal{D}$ output is still legitimate and we can wait for S' output to change. If $Q \cap P'_1.S'(t_{1,0}) = \emptyset$, we can set the S output on processes in $P'_1.S'(t_{1,0})$ to an arbitrary subset of P_2 that is not equal to $P'_1.S'(t_{1,0})$. Since $|P_2| \geq 2$, this is always possible. Now P_2 becomes the live component in $\Pi\Omega\Upsilon_k$, and the $\mathcal{D}$ output becomes legitimate. Then we wait for a time $t'_{1,0} > t_{1,0} + 1$ at which P'_1 becomes quasi-live again, and turn to run $R_{1,1}$.

Using a similar procedure in part 1, and the same process-crashing technique described in $R_{1,0}$, we can construct other runs $R_{j,m}$.

We first claim that we will never reach the runs $R_{j,m}$ such that $P'_j \cap Q = \emptyset$. Otherwise, we know that *lbound*$'(P'_i) \geq |P'_i|$ for all $i \in \{1, \ldots, j-1\}$. Based on the component sorting of π', we have $Q \subseteq \bigcup_{i=1}^{j-1} P'_i$ This implies $\sum_{i=1}^{j-1} \geq k$, contradictory to the (Ω1) requirement of $\Pi\Omega\Upsilon_{k-1}$. Second we claim that there exist a $j \in \{1, \ldots, s\}$ such that at any time t, *lbound*$'(P'_j, t) < |$*lbound*$'(P'_j, t)|$ in all runs $R_{j,m}$. Otherwise we will have $\sum_{i=1}^{j-1} = n + 1 > k - 1$. Let R^∞ be a infinite run such that every $R_{j,m}$ shares a prefix with it until $t_{s,m}$. It's obvious in R^∞, we do not crash any process in P'_j. Since $P'_j \cap Q \neq \emptyset$ and $P_1 = Q$ is always the live component for $\Pi\Omega\Upsilon_k$, our $\mathcal{D}$ output is legitimate. But T cannot provide a stable Υ output S' for $\Pi\Omega\Upsilon_{n-1}$.

Part 2. In this part, $k = n$, and we let D to output the following initially: (a) The processes are partitioned into two components P_1 and P_2 such that $P_1 = Q$ and $P_2 = P \setminus Q = \{p_{n+1}\}$. (b) All processes in P_1 set their *lbound* and S to 1 and $\{p_1\}$. (c) p_{n+1} set its *lbound* and S to 0 and $\emptyset$.

Run R_0: We let all process in P to run until T outputs *cid*$'$. This is possible because the initial output of D is valid for $\Pi\Omega\Upsilon_n$. Suppose at time t_0, every process outputs their *cid*$'$, and $\pi' = \{P'_1, P'_2, \ldots, P'_s\}$ is the partition derived from the outputs. We define *lbound*$'(P'_j, t)$ similarly as in Lemma 9.

There must be a component in π' which contains process p_{n+1}. Without loss of generality, we assume $p_{n+1} \in P'_s$. We then set *lbound* and S values of processes based on the two possible cases: (1) if $P'_s = P_2$, we set *lbound* and S of processes in Q to n and $\emptyset$, respectively, and set *lbound* of p_{n+1} to 0 and S output of

p_{n+1} to $\emptyset$; (2) if $P'_s \neq P_2$, we set *lbound* and S of processes in Q to $n-1$ and $Q \cap P'_s$, respectively, and set *lbound* of p_{n+1} to 1 and S output of p_{n+1} to $\emptyset$. Note that the *lbound* output values always satisfy that the sum of maximum *lbound* values of the two components P_1 and P_2 is at most n, i.e., satsifying ($\Pi\Omega 1$) of $\Pi\Omega\Upsilon_n$. In the following, we design a set of runs $R_{j,m}$ for $1 \leq j < s$ and $m \geq 0$.

Run $R_{1,0}$: R_1 runs exactly the same as R_0 until time t_0. Let $t_1 = t_0$. If $\mathit{lbound}'(P'_1, t_1) \geq |P'_1|$, we turn to run $R_{2,0}$. Otherwise, we crash all processes in $P \setminus P'_1$ at time $t_1 + 1$, and let processes in P'_1 run. Since $P'_1 \neq \emptyset$ and $P'_1 \subseteq Q$, our $\mathcal{D}$ output is still legitimate for $\Pi\Omega\Upsilon_n$. Then we wait for a time $t_{1,0} > t_1 + 1$ at which P'_1 becomes quasi-live. A component P'_j is quasi-live at time t iff. $\forall p, q \in P'_j(H'(p,t).\mathit{lbound}' = H'(q,t).\mathit{lbound}' \wedge H'(p,t).S' = H'(q,t).S' \neq P'_j \backslash F(t)$ and $\mathit{lbound}'(P'_j, t) \geq |P'_j \backslash F(t)| \vee H'(p,t).S' \neq \emptyset$. P'_j is quasi-live always implies the S' output stables. In the following we will use $P'_j.S'(t)$ as the short form to the stable S' output at time t on processes in P'_j. Since P'_1 is the only component having correct process, $t_{1,0}$ must exist. Since we have not crashed any process in P'_1 yet, $\mathit{lbound}'(P'_1, t_{1,0}) \geq |P'_1 \setminus F(t_{1,0})|$ implies $\mathit{lbound}'(P'_1, t_{1,0}) \geq |P'_1|$. If $\mathit{lbound}'(P'_1, t_{1,0}) \geq |P'_1|$, we turn to run $R_{2,0}$. Otherwise, $P'_1.S'(t_{1,0})$ must be a non-empty set. We crash all process in $P'_1 \setminus P'_1.S'(t_{1,0})$ at time $t_{1,0}+1$. Since $P'_1.S'(t_{1,0}) \neq \emptyset$, at least one process in Q does not crash and H is still legitimate. Then we wait for a time $t'_{1,0} > t_{1,0} + 1$ at which P'_1 becomes quasi-live again, and turn to run $R_{1,1}$.

Run $R_{j,m}$ ($1 \leq j < s$**,** $m \geq 1$): $R_{j,m}$ runs exactly the same as $R_{j,m-1}$ until $t_{j,m-1}$. At time $t_{j,m-1}+1$, instead of crashing processes in $P'_j.S'(t_{j,m-1})$, we suppress the execution of these processes and schedule the other processes to take exactly the same sequence of steps as in $R_{j,m-1}$ until $t'_{j,m-1}$. After $t'_{j,m-1}$, we allow all processes in P'_j to run until a time $t_{j,m}$ such that (1) every process in P'_j takes at least one step between $t'_{j,m-1}$ and $t_{j,m}$; and (2) P'_j becomes quasi-live at time $t_{j,m}$. Since at time $t_{j,m}$, we have not crashed any process in P'_j yet, $\mathit{lbound}'(P'_j, t_{j,m}) \geq |P'_j \setminus F(t)|$ still implies $\mathit{lbound}'(P'_j, t_{j,m}) \geq |P'_1|$. If $\mathit{lbound}'(P'_j, t_{j,m}) \geq |P'_j|$, we turn to run $R_{j+1,0}$. Otherwise, we crash all process not in $P'_j.S'(t_{j,m})$ at time $t_{j,m}+1$. Since $P'_j.S'(t_{j,m}) \neq \emptyset$, at least one process in Q does not crash and H is still legitimate. Then we wait for a time $t'_{j,m} > t_{j,m} + 1$ at which P'_j becomes quasi-live again, and turn to run $R_{j,m+1}$.

Run $R_{j,0}$ ($2 \leq j \leq s$): $R_{j,0}$ runs exactly the same as $R_{j-1,0}$ until time t_{j-1}. We know components P'_i ($1 \leq i < j-1$) has been crashed at time t_{j-1} with $\mathit{lbound}'(P'_i, t_{j-1}) \geq |P'_i|$. If $\mathit{lbound}'(P'_{j-1}, t_{j-1}) \geq |P'_{j-1}|$, let $t_j = t_{j-1}+1$. Otherwise, it must be true that a run $R_{j-1,m_{j-1}}$ exists such that $\mathit{lbound}'(P'_{j-1}, t_{j-1,m_{j-1}}) \geq |P'_{j-1}|$. At time $t_{j-1}+1$ instead of crashing processes in $P'_{j'}$ ($j' \geq j$), we suppress these process from running and allow the processes in P'_{j-1} to take exactly the same sequence of steps as in run $R_{j-1,m_{j-1}}$ until time $t_{j-1,m_{j-1}}$. After time $t_{j-1,m_{j-1}}$, we allow every process in $P'_{j'}$ ($j' \geq j-1$) to run. Let $t_j > t_{j-1,m_{j-1}}$ be the time such that everyone in $P'_{j'}$ takes at least one step by $t_j - 1$ and after $t_{j-1,m_{j-1}}$. At time t_j we further crash processes in component P'_{j-1}. Now we have crash all processes in P'_i ($1 \leq i \leq j-1$). Since we have $\mathit{lbound}'(P'_i, t_j)) \geq |P'_i|$ and $\sum_{i=1}^{j-1} \mathit{lbound}'(P_i) \leq k-1$, $|Q| = k$ implies at least one process in Q is not crashed. So H is still a legitimate.

Now we look at component P'_j. If $\mathit{lbound}'(P'_j, t_j) \geq |P'_j|$, we turn to run $R_{j+1,0}$. Otherwise, we crash all processes in $P'_{j'}$ ($j < j' \leq s$) at time $t_j + 1$ and let processes in P'_j run. Because $P'_j \subseteq Q$ and $P'_j \neq \emptyset$ implies at least one process in Q is in P'_j, H is still a legitimate $\Pi\Omega\Upsilon_n$ output in this run. Then we wait for a time $t_{j,0} > t_j + 1$ at which P'_j becomes quasi-live. Since P'_j is the only component having correct process, $t_{j,0}$ must exist. If $\mathit{lbound}'(P'_j, t_{j,0}) \geq |P'_j|$, we turn to run $R_{j+1,0}$. Otherwise, we crash all process not in $P'_j.S'(t_{j,0})$ at time $t_{j,0}+1$. Since $P'_j.S'(t_{j,0}) \neq \emptyset$, at least one process in Q does not crash and H is still legitimate in this run. We wait for a time $t'_{j,0} > t_{j,0} + 1$ at which P_j becomes quasi-live again, and turn to run $R_{j,1}$.

In the above, we have constructed a series of runs, in which our H is always a legitimate output of $\Pi\Omega\Upsilon_n$. If $R_{s,0}$ is never entered, there must exist a $j \in \{1, \ldots, s-1\}$, such that $\mathit{lbound}'(P'_j, t_j)) < |P'_j|$ in run $R_{j,0}$ and *lbound*$'(P'_j, t_{j,m}) < |P'_j|$ for all $m \geq 0$ in run $R_{j,m}$. Let R^∞ be the infinite run such that every $R_{j,m}$ shares a prefix with it until $t_{j,m}$. In this run, the S' output on processes in P'_j cannot stabilize. So, T is not a correct transformation algorithm. If otherwise, we enter run $R_{s,0}$, we construct a series of run $R_{s,m}$ in the following.

Run $R_{s,0}$: According to the runs $R_{j,m}$ ($j \in \{1, \ldots, s-1\}$), $R_{s,0}$ being entered implies that there exist a run $R_{s-1,m}$ and a time t such that *lbound*$'(P'_j, t) \geq |P'_j|$ for all $j \in \{1, \ldots, s-1\}$ in $R_{s-1,m}$. If $P'_s = P_2$, we know $\bigcup_{j=1}^{s-1} P'_j = Q$, thus $\sum_{j=1}^{s-1}$ *lbound*$'(P'_j) = n$. This contradicts to the fact that $\sum_{j=1}^{s-1}$ *lbound*$'(P'_j) \leq n-1$. So we know $P'_s \neq P_2$ and *lbound*$(P_1, t) = n-1$.

$R_{s,0}$ runs exactly the same as $R_{s-1,m}$ until $t_{s-1,m}$. At time $t_{s-1,m}+1$, we crash all processes not in P'_s. For each process in $Q \cap P'_s$, we also set its S output to $\emptyset$. Then we let processes in P'_s run. In this run, P_2 is a live component with respect to $\Pi\Omega\Upsilon_n$, so the $\mathcal{D}$ output is still legitimate. Now P'_s is the only component having correct processes, so T must give a legitimate output for $\Pi\Omega\Upsilon_{n-1}$ on processes in P'_s. Because *lbound*$'(P'_s) < |P'_s|$ (otherwise $\sum_{j=1}^{s}$ *lbound*$'(P'_j) = n+1 > n-1$), we know T must provide a non-empty correct Υ output S' on every correct process. Suppose at time $t_{s,0}$, S' becomes stable on all processes in P'_s. Let the stable $S_0 = P'_s.S'(t_{s,0}) \neq \emptyset$. At time $t_{s,0}+1$ we crash all process not in S_0. Now S_0 becomes the exact set of correct processes. If $p_{n+1} \in S_0$, our H is still legitimate, so we can just wait for S' on processes in S_0 to change. If $p_{n+1} \notin S_0$, it must be true that $S_0 \subseteq Q$. If $S_0 \neq Q$, we can set S output of processes in S_0 to $Q \setminus S_0$. Otherwise, we set S to $\{p_1\}$. Since $|Q| = n = k \geq 2$, $S \neq S_0$. Now $P_1 = Q$ becomes a live component regarding to $\Pi\Omega\Upsilon_n$. So we can still wait for S' to change. Suppose S' becomes stable at time $t'_{s,0}$ and $P'_s.S'(t'_{s,0}) \neq S_0$.

Now we can construct run $R_{s,1}$, $R_{s,2}$, ... in a similar way using the same technique in $R_{j,m}$. And we let R^∞ be the infinite run such that every $R_{s,m}$ shares a prefix with it until $t_{s,m}$. It's obvious that in R^∞, we do not crash any process in P'_s. So P_2 is a live component in $\Pi\Omega\Upsilon_n$ and our $\mathcal{D}$ output is legitimate. But T cannot provide a stable Υ output S' for $\Pi\Omega\Upsilon_{n-1}$.

Our proof in part 1 and part 2 concludes that $\Pi\Omega\Upsilon_k$ cannot be transformed to $\Pi\Omega\Upsilon_{k-1}$. □

Lemma 12 *For any $n \geq 3$, (1) $\Pi\Upsilon$ cannot be transformed into $\Pi\Omega\Upsilon_{n-3}$, and (2) $\Pi\Upsilon$ cannot be transformed into $\Pi\Omega\Upsilon_{n-2}$ when n is odd.*

Proof. When $n \leq 2$, by Lemma 4 and Lemma 7, we know $\Pi\Upsilon$ can be transformed into $\Upsilon = \Pi\Omega\Upsilon_0$, so $\Pi\Upsilon$ can be transformed into $\Pi\Omega\Upsilon_{n-2}$ when $n = 2$.

For the case when $n \geq 3$, we firstly prove $\Pi\Upsilon$ cannot be transformed into $\Pi\Omega\Upsilon_{n-3}$. Suppose, for a contradiction, that we have an algorithm T that transforms any failure detector $\mathcal{D}$ in $\Pi\Upsilon$ to a failure detector $\mathcal{D}'$ in $\Pi\Omega\Upsilon_{n-3}$. Let (S, cid) denote the output of $\mathcal{D}$, and $(S', \mathit{lbound}, \mathit{cid}')$ denote the output of $\mathcal{D}'$ generated by algorithm T.

We consider the partition of P, $\pi = \{P_1, P_2, \cdots, P_s\}$ where $s = \lceil \frac{n}{2} \rceil$ and $P_1 = \{p_1, p_2\}, P_2 = \{p_3, p_4\}, \cdots, P_s = \{p_n, p_{n+1}\}$ if n is odd and $P_k = \{p_{n-1}, p_n, p_{n+1}\}$ if n is even. Firstly, let all processes are correct processes. Their *cid* outputs correspond to our partition construction, and S outputs are arbitrary. Since there exist two components P_i and P_j containing correct process, the output of $\mathcal{D}$ satisfies the specification of $\Pi\Upsilon$. Then we let T run to some time t_0 at which the outputs $(S', \mathit{lbound}, \mathit{cid}')$ of $\mathcal{D}'$ are generated by T and $\mathit{cid}' \neq \perp$ for all processes. By $\Pi C1$, all correct processes eventually have $\mathit{cid}' \neq \perp$, so such time t_0 exists. From the cid' outputs, it is clear how processes are partitioned with respect to $\mathcal{D}'$. Suppose the partition of $\mathcal{D}'$ is $\pi' = \{Q_1, Q_2, \cdots, Q_{s'}\}$. We define $Q_i.\mathit{lbound}(t) = \max\{H'(p, t').\mathit{lbound} \mid t' \leq t, p \in Q_i \setminus F(t')\}$. At time t, we say that component Q_i is *overflowing* if $Q_i.\mathit{lbound}(t) \geq |Q_i|$, and we say Q_i is *quasi-live* if $Q_i.\mathit{lbound}(t) < |Q_i|$ and all processes in

Q_i has the same nonempty output S' which is not the set of correct processes in Q_i. By ($\Pi\Upsilon 1$), for a live component Q_i, there exists a time after which Q_i is either overflowing or quasi-live. We construct run R using the following procedure to reach a contradiction.

Procedure: At any time t, if we find overflowing component Q_i, we crash all processes in Q_i at time $t+1$ and repeat the procedure again. Otherwise, since any survive component Q_j is not overflowing, we can find some quasi-live components eventually. Suppose, at time t', all quasi-live components are $\{Q_{i_1}, Q_{i_2}, \cdots, Q_{i_l}\}$ where $l \geq 1$. From time $t'+1$, we suppress all processes p such that $p \notin Q_{i_j}$ for any j or $p \in Q_{i_j}$ and $p \notin H'(p,t').S'$. Since S' output in Q_{i_1} is not empty, at least one process is not suppressed. Intuitively, we try to simulate a run R' in which all suppressed processes are also crashed. In run R', for each correct process p (p is neither crashed nor suppressed in run R), we can choose suitable non-empty S output for it. This is because in our construction of partition $\pi = \{P_1, P_2, \cdots, P_s\}$, each component has at least two processes. But in run R', every components generated by T is not the live components at time $t'+1$, so, eventually, at least one correct process p changes its S' output. Since p cannot distinguish run R and run R', it also changes its S' output in run R. After it, we suppress all processes in $Q_{i_j} (1 \leq j \leq l)$ if Q_{i_j} contains some process which changes its S' output in run R. If there still exists some process which has not been suppressed, we continue run R. By the same argument, eventually, all correct processes in run R are suppressed. This means all components $Q_{i_j} (1 \leq j \leq l)$ contain at least one process which changes its S' output after time t'. Then, we recover all processes which does not crash in run R (that is, the processes which are not in the overflowing component) and let them take at least one step.

By ($\Pi\Omega 2$), sum of *lbound* of all components are not greater than $n-3$, so there are at least 4 processes which are not crashed in run R. So, we can repeat the above procedure infinitely many times. Then, (1) we firstly define a failure pattern F for run R which describe above, then prove that (2) all correct processes in R take an infinite number of steps; (3) the output (S, cid) of $\mathcal{D}$ satisfies the specification of $\Pi\Upsilon$; (4) the output $(S', \mathit{lbound}, \mathit{cid}')$ of $\mathcal{D}'$ violates the specification of $\Pi\Omega\Upsilon_{n-3}$. Then, by (2)(3), we know run R is a legitimate run of algorithm T under some specified failure pattern F and we can reach the contradiction with (4).

(1): We define $F(t) = \{p |$ let $p \in Q_i, Q_i.\mathit{lbound}(t-1) \geq |Q_i|\}$. Firstly, by the definition of $Q_i.\mathit{lbound}(t)$, if $p \in F(t)$, then $p \in F(t'), \forall t' \geq t$. Thus, for any component Q_j, either all processes in Q_j are correct processes or none of them are correct.

(2): For any correct process in R, since it is not in the overflowing component at any time, it does not crash in our procedure. So, it takes at least one step in the end of the procedure. Thus, every correct process takes an infinite number of steps.

(3): ($\Pi C1$) and ($\Pi C2$) hold directly from our construction of partition. Since there are at least 4 correct processes in run R. Then, by our construction of partition $\pi = \{P_1, P_2, \cdots, P_s\}$, we know at least two components P_i and P_j contains correct processes. Thus, ($\Pi C3$) holds.

(4): Prove by contradiction. If the output $(S', \mathit{lbound}, \mathit{cid}')$ of $\mathcal{D}'$ satisfies the specification of $\Pi\Omega\Upsilon_{n-3}$, suppose Q_j is the live component. Then, $Q_j.\mathit{lbound}(t) < |Q_j|$ for any t, otherwise, Q_j does not contain correct process by our definition of failure pattern. Thus, eventually, all correct processes in Q_j output the same nonempty $S' \subseteq Q_j$ such that S' is not the set of correct processes in Q_j. Suppose this time is t_j, then after t_j, Q_j is quasi-live. By the procedure, there exists the time t'_j, at least one correct process in Q_j changes its S' output. This means the S' output in Q_j is not stable, which violates ($\Pi\Upsilon 1$)

In fact, we can easily check that the same argument can also prove $\Pi\Upsilon$ cannot be transformed into $\Pi\Omega\Upsilon_{n-2}$ when n is an odd number. In this case, all components P_i contain exactly two processes, and since at least 3 processes are not crashed by the procedure, we have at least two components with correct processes left. $\square$