Partition Approach to Failure Detectors for k-Set Agreement

Wei Chen[†] Jialin Zhang[‡] Yu Chen[†] Xuezheng Liu[†]

[†]Microsoft Research Asia
{weic,ychen,xueliu}@microsoft.com

[‡]Tsinghua University
zhanggl02@mails.tsinghua.edu.cn

Abstract

In k-set agreement problem, every process proposes a value and eventually at most k different values can be decided. When $k > 1$, different subset of processes may decide on different values, and thus it naturally exhibits partition among processes based on their decision values.

In this paper, we propose the partition approach to define failure detectors that capture the partition nature of k-set agreement. The power of the partition approach is to further weaken failure detectors that are already very weak in solving k-set agreement, and thus invalid the failure detectors as candidates for the weakest failure detectors for k-set agreement. Using the approach, we propose two new classes of failure detectors, statically partitioned failure detectors Π_k and splittable partitioned failure detectors Π_k^S, both are strong enough to solve k-set agreement in the message passing model. However, we show that Π_k is strictly weaker than Ω_k, the weakest failure detectors known so far for k-set agreement, and Π_k^S is even weaker than Π_k. The partition approach provides a new dimension to weaken failure detectors related to k-set agreement. It is an effective way to check whether a failure detector is the weakest one solving k-set agreement or not. Together with [4], we show that so far all candidates for the weakest failure detectors including Ω_k and Υ in both the message-passing model and the shared-memory model have failed our partition test.

MSR-TR-2007-49

1 Introduction

The problem of k-set agreement is introduced in [3] as a generalization of the consensus problem (the case of $k = 1$). In k-set agreement, each process from a set of $n > k$ processes 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 some correct process (process that does not crash in the run) decides.[1]

It has been shown that k-set agreement cannot be solved in asynchronous systems when k or more processes may crash [1, 10, 18]. Several studies have introduced various failure detector classes to circumvent the impossibility result [19, 15, 9, 13, 14]. In particular, in [14] Mostefaoui et.al. study the relationship among these failure detector classes and show that class Ω_k is the weakest among them to solve k-set agreement. Ω_k is an extension to leader elector Ω, which has been shown to be the weakest failure detector solving consensus [2]. A failure detector in Ω_k outputs a set of at most k processes, and eventually all correct processes output the same set of processes that contains at least one correct process. However, it is shown in [14] that a majority of correct processes is needed to solve k-set agreement using Ω_k.

In [7], Delporte-Gallet et.al. show that the above majority requirement can be generalized to a quorum failure detector Σ, which outputs a set of processes called quorum such that: ($\Sigma 1$) any two quorums intersect; and ($\Sigma 2$) eventually all quorums contain only correct processes. Combining the result in [2], they show that $\Omega \times \Sigma$ is the weakest class of failure detectors solving consensus in any environment.

Therefore, among the known failure detectors, $\Omega_k \times \Sigma$ is the weakest class of failure detectors solving k-set agreement in the message-passing model when $k > 1$. Ω_k was further conjectured in [17] to be the weakest failure detector for k-set agreement among all possible failure detectors in the shared-memory model.

In k-set agreement when $k > 1$, since different processes may decide on different values, it naturally exhibits partition among processes based on their decision values. In this paper, we propose a new approach call the *partition approach* that effectively weakens existing failure detectors by capturing the partition nature of k-set agreement. Roughly speaking, 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. The safety properties guarantee that collectively all components decide on at most k different values, while the liveness properties guarantee that eventually some decision is made in the live component. Since those processes in non-live components may generate quite arbitrary failure detector outputs, intuitively the partitioned failure detectors are weaker than existing ones that require both safety and liveness on all processes.

By applying the partition approach to $\Omega_k \times \Sigma$, we propose two partitioned failure detectors Π_k and Π_k^S in asynchronous message passing systems, and show that they are strong enough to solve k-set agreement but are strictly weaker than $\Omega_k \times \Sigma$.

To start, we select the failure detector class $\Omega_k'' \times \Sigma$ instead of $\Omega_k \times \Sigma$ as the basis of non-partitioned failure detectors. In [5] we define Ω_k'' and show that it is equivalent to Ω_k. Failure detectors in Ω_k'' outputs $(\mathit{isLeader}, \mathit{lbound})$, where *isLeader* is a Boolean variable indicating whether a process itself is a leader, and *lbound* is a number that estimates the upper bound of the number of leaders in the system (see Section 2 for its definition). The reason we choose Ω_k'' instead of Ω_k is because the outputs of Ω_k'' do not refer to other processes as the case of Ω_k, so it is cleaner to extend it to the partitioned envi-

[1]In asynchronous systems with reliable channels, a correct process that decides can send out its decision value to all processes so that all correct processes eventually decide. Therefore, our Termination property implies a different version that requires all correct processes eventually decide.

$$\begin{tikzcd}
 & & & \Omega''_1 \times \Sigma \arrow[dl] \arrow[dr, shift left] \arrow[dr, leftarrow, shift right] & & \\
 & & \Omega''_2 \times \Sigma \arrow[dr] \arrow[dddl, dotted, no head] & & \Pi_1 \arrow[dl] \arrow[dr, shift left] \arrow[dr, leftarrow, shift right] & \\
 & & & \Pi_2 \arrow[dr] \arrow[dddl, dotted, no head] & & \Pi^S_1 \arrow[dl] \\
 & & & & \Pi^S_2 \arrow[dddl, dotted, no head] & \\
 & \Omega''_{k-1} \times \Sigma \arrow[dl] \arrow[dr] & & & & \\
\Omega''_k \times \Sigma \arrow[dr] & & \Pi_{k-1} \arrow[dl] \arrow[dr] & & & \\
 & \Pi_k \arrow[dr] & & \Pi^S_{k-1} \arrow[dl] & & \\
 & & \Pi^S_k & & &
\end{tikzcd}$$

(a) in the purely asynchronous model $\mathcal{A}$ ($n \geq 2k-1$ is required to show that the transformations from $\Omega''_k \times \Sigma$ to Π^S_{k-1} and stronger classes are impossible).

$$\begin{tikzcd}
 & & & \Omega''_1 \arrow[dl] \arrow[dr, shift left] \arrow[dr, leftarrow, shift right] & & \\
 & & \Omega''_2 \arrow[dr] \arrow[dddl, dotted, no head] & & \Pi_1 \arrow[dl] \arrow[dr, shift left] \arrow[dr, leftarrow, shift right] & \\
 & & & \Pi_2 \arrow[dr] \arrow[dddl, dotted, no head] & & \Pi^S_1 \arrow[dl] \\
 & & & & \Pi^S_2 \arrow[dddl, dotted, no head] & \\
 & \Omega''_{k-1} \arrow[dl] \arrow[dr] & & & & \\
\Omega''_k \arrow[dr] & & \Pi_{k-1} \arrow[dl] \arrow[dr] & & & \\
 & \Pi_k \arrow[dr] & & \Pi^S_{k-1} \arrow[dl] & & \\
 & & \Pi^S_k & & &
\end{tikzcd}$$

(b) in the asynchronous models $\mathcal{A}[\Sigma]$ or $\mathcal{A}[\mathit{Maj}]$ with $n \geq 4k+2$.

Figure 1: Relationship lattices of paritioned failure detectors. All failure detector classes in the lattices can be used to solve k-set agreement in their corresponding models.

ronment without worrying whether a failure detector output refers to processes in the same partitioned component.

Our first step is to apply static partitions to $\Omega''_k \times \Sigma$, which leads to the definition of *statically partitioned failure detectors* Π_k (Section 3.1). Informally, processes are statically partitioned into multiple components at the beginning of the run. Each component P_i has a failure detector with all the safety properties of $\Omega''_{k_i} \times \Sigma$ resticted to P_i. In particular, all quorum outputs are contained within P_i. The sum of k_i's is at most k. Moreover, at least one component P_j also satisfies all liveness properties of $\Omega''_{k_j} \times \Sigma$. The intuition behind is that with Π_k each component may decide at most k_i values by the safety requirement so together there are at most k decisions, while at least one component can eventually decide by the liveness requirement.

Next we further weaken Π_k by allowing dynamic splitting of components during the run. This leads to the definition of Π^S_k, which we call *splittable partitioned failure detectors* (Section 3.2). Its definition is not entirely straightforward due to its flexibility.

With the new families of failure detectors $\{\Pi_z\}_{1 \leq z \leq k}$, and $\{\Pi^S_z\}_{1 \leq z \leq k}$, we compare their strengths with $\{\Omega''_z \times \Sigma\}_{1 \leq z \leq k}$ (Section 4). Based on a siginificant amount of proof work, we summarize their relationship in purely asynchronous model $\mathcal{A}$ with a nice lattice structure shown in Figure 1 (a). In the lattice, each arrow from class $\mathcal{C}_1$ to $\mathcal{C}_2$ represents that $\mathcal{C}_1$ has enough information to be transformed into $\mathcal{C}_2$ (i.e., $\mathcal{C}_1$ is at least as strong as $\mathcal{C}_2$), and if class $\mathcal{C}_1$ has no directed path to class $\mathcal{C}_2$, there is no transformation from $\mathcal{C}_1$ to $\mathcal{C}_2$. Several important results are summarized by the lattice. First, as we expected Π_k weakens $\Omega''_k \times \Sigma$ while Π^S_k further weakens Π_k. Second, even failure detectors in Π_2 with just two components is not strong enough to be transformed into $\Omega''_k \times \Sigma$, and even failure detectors in Π^S_2 with only one dynamic split is not strong enough to be transformed into Π_k. This shows that partitioning and dynamic splitting are indeed efficient techniques that weaken failure detectors. Third, for all $z \geq 2$, none of the classes $\Omega''_z \times \Sigma$, Π_z, and Π^S_z can be transformed into $\Omega''_{z-1} \times \Sigma$, Π_{z-1}, or Π^S_{z-1}. In fact, using a result in [14] we further show that $\Omega''_z \times \Sigma$, Π_z, and Π^S_z are not strong enough to solve $(z-1)$-set agreement.

Furthermore, we look into stronger system models $\mathcal{A}[\Sigma]$, which are the asynchronous model $\mathcal{A}$ augmented with quorum failure detector Σ. We would like to see if adding a global quorum system Σ would collapse the lattice structure. Surprisingly we find out that when $n \geq 4k+2$, the same lattice structure still holds (Figure 1 (b)) for $\mathcal{A}[\Sigma]$ (and even the stronger $\mathcal{A}[\mathit{Maj}]$ in which a majority of processes are correct). This means that static partitioning and dynamic splitting still weaken the strength of failure detectors in $\mathcal{A}[\Sigma]$ and $\mathcal{A}[\mathit{Maj}]$. Our explanation is that partitioning not only weakens the global quorum systems but also weakens the leader election part Ω''_k. This is confirmed with additional results in our paper not covered by the relationship lattices. The direct implication is that, even in $\mathcal{A}[\Sigma]$ or $\mathcal{A}[\mathit{Maj}]$, Ω''_k (or Ω_k) is not the weakest failure detector class solving k-set agreement.

Finally, we show that all failure detector classes in the relationship lattices can be used to solve k-set agreement (Section 5). To do so, we provide an algorithm using the weakest failure detector class Π^S_k in the lattices and prove that it solves k-set agreement in model $\mathcal{A}$. The algorithm is a further extension to the algorithm we presented in [5], which is an extension to the Paxos algorithm [11] to solve k-set agreement with Ω''_k in model $\mathcal{A}[\mathit{Maj}]$.

To summarize, our contributions are: (a) we propose the use of the partition approach to weaken failure detectors for k-set agreement, and define two classes of failure detectors Π_k and Π^S_k using this approach; (b) we fully characterize the relationship between Π_k, Π^S_k and Ω_k and prove that Π_k is strictly weaker than Ω_k while Π^S_k is strictly weaker than Π_k, and we show that the same relationship holds even in stronger models $\mathcal{A}[\Sigma]$ and $\mathcal{A}[\mathit{Maj}]$; and (c) we present an algorithm that solves k-set agreement using the weakest failure detector class Π^S_k we defined. Our results deepen the understanding of the k-set agreement problem and its associated failure detectors, and they contribute to the pursuit of the weakest failure detectors for k-set agreement.

The partition approach is applicable to other failure detectors in other system model. In fact, in [4] we apply the partition approach to discover a series of new failure detectors in the shared memory model that are even weaker than Υ, a recently proposed failure detector in [8] that is the weakest one ever found to solve any impossible decision task in the shared-memory model before our work. We believe that the partition approach opens a new dimension to weaken failure detectors for k-set agreement, and it is an effective test to check whether or not a failure detector is a weakest one for k-set agreement. So far all candidates for the weakest failure detectors including Ω_k and Υ in both the message-passing model and the shared-memory model have failed our partition test.

Moreover, partitioned failure detectors match the system environments that allow partitions and thus can lead to potential implementations in real systems. In Section 6, we discuss some of the future directions of this work as our conclusion.

The main text of this report focuses on the definitions and explanations of the main results. All detailed proofs of the results are included in the appendix.

2 Model

We consider asynchronous message passing distributed systems augmented with failure detectors. Our formal model is the same as the model in [2]. We only provide informal description due to space contraints.

We consider a system with n ($n > k$) processes $P = \{p_1, p_2, \ldots, p_n\}$. Let $\mathcal{T}$ be the set of 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. Let $\mathit{correct}(F)$ denote the set of *correct processes*, those that do not crash in F. 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.

We now define Ω''_k, whose output is $(\mathit{isLeader}, \mathit{lbound})$. We say that a process p is an *eventual leader* if p is correct and there is a time after which p's *isLeader* outputs are always *True*.[2] Failure detectors in Ω''_k satisfy the following properties: ($\Omega''1$) the *lbound* outputs never exceed k; ($\Omega''2$) eventually, the *lbound* outputs of all

[2]The definition of eventual leader also applies to the failure detector classes Π_k and Π^S_k defined later.

processes do not change and are the same; ($\Omega''3$) eventually the *isLeader* outputs on any correct process do not change; ($\Omega''4$) there is at least one eventual leader; and ($\Omega''5$) the number of eventual leaders is eventually bounded by the *lbound* outputs.

Processes communicate with each other by sending and receiving messages over communication channels, which are available between every pair of processes. Channels are reliable in that it does not create or duplicate messages, and any message sent to any correct process is eventually received.

A deterministic algorithm A using a failure detector $\mathcal{D}$ executes by taking *steps*. In each step, a process p first receives a message (it could be a null message), queries its failure detector module, then changes its local state and sends out a finite number of messages to other processes. 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 $\mathcal{D}$ is an infinite sequence of steps such that (a) every correct process takes an infinite number of steps, and (b) every message sent to a correct process is eventually received.

We say that a failure detector class $\mathcal{C}_1$ is *weaker than* a failure detector class $\mathcal{C}_2$ in a model $\mathcal{M}$, if there is a transformation algorithm T in model $\mathcal{M}$ such that using any failure detector in $\mathcal{C}_2$, algorithm T implements a failure detector in $\mathcal{C}_1$. In this case, we denote it as $\mathcal{C}_1 \preceq \mathcal{C}_2$ and also refer to it as $\mathcal{C}_2$ can be transformed into $\mathcal{C}_1$. We say that $\mathcal{C}_1$ is *strictly weaker than* $\mathcal{C}_2$ if $\mathcal{C}_1 \preceq \mathcal{C}_2$ and $\mathcal{C}_2 \not\preceq \mathcal{C}_1$. We say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are equivalent if $\mathcal{C}_1 \preceq \mathcal{C}_2$ and $\mathcal{C}_2 \preceq \mathcal{C}_1$.

We consider three asynchronous system models. In the purely asynchronous model $\mathcal{A}$, there is no bound on the delay of messages and the delay between steps that a process takes. Model $\mathcal{A}[\Sigma]$ is $\mathcal{A}$ added with Σ, and model $\mathcal{A}[Maj]$ is $\mathcal{A}$ with the requirement that a majority of processes are correct. It is straightforward to see that $\mathcal{A}[Maj]$ is stronger than $\mathcal{A}[\Sigma]$ in that Σ can be implemented in $\mathcal{A}[Maj]$.

3 Specification of Partitioned Failure Detectors

In this section, we present the formal specifications of two new classes of failure detectors Π_k and Π_k^S, both of which are based on the idea of process partition. We also explain the intuitions behind their definitions.

3.1 Statically partitioned failure detectors Π_k

The statically partitioned failure detectors Π_k formalize the idea of static partitions. The output of a failure detector $\mathcal{D}$ in Π_k for process p is a tuple $(isLeader, lbound, Quorum)$, where *isLeader* is a Boolean value indicating whether this process is a leader, *lbound* is a non-negative integer indicating the upper bound on the number of possible leaders in p's partitioned component, and $Quorum \subseteq P$.

A *partition* of P is $\{P_1, \ldots, P_s\}$, 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 $P[p]$ to denote the partitioned component that contains p. For a component $P_j \subseteq P$, a failure pattern F and a failure detector history H, we define $lbound(P_j, F, H) = \max\{H(p,t).lbound \mid t \in \mathcal{T}, p \in P_j \setminus F(t)\}$,[3] and $Leaders(P_j, F, H) = \{p \in P_j \cap correct(F) \mid \exists t, \forall t' > t, H(p,t').isLeader = True\}$. We usually use $lbound(P_j)$ and $Leaders(P_j)$ and omit F and H since they are clear from the context. The value $lbound(P_j)$ is the maximum *lbound* value among processes in component P_j, while $Leaders(P_j)$ is the set of eventual leaders in P_j.

A failure detector $\mathcal{D}$ is in the class Π_k if for any failure pattern F and any failure detector history $H \in \mathcal{D}(F)$, there exists a partition of P, $\{P_1, \ldots, P_s\}$, such that H satisfies the following set of safety and liveness properties. The safety properties are:

($\Pi\Sigma1$) The quorum output of a process p is al-

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

ways contained within p's partitioned component. Formally, $\forall t \in \mathcal{T}, \forall p \notin F(t), H(p,t).\mathit{Quorum} \subseteq P[p]$.

($\Pi\Sigma 2$) The quorum outputs in the same partitioned component always intersect. Formally, $\forall t_1, t_2 \in \mathcal{T}, \forall p_1 \notin F(t_1), \forall p_2 \notin F(t_2), P[p_1] = P[p_2] \Rightarrow H(p_1,t_1).\mathit{Quorum} \cap H(p_2,t_2).\mathit{Quorum} \neq \emptyset$.

($\Pi\Omega 1$) 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:

($\Pi\Sigma 3$) Eventually the quorum outputs by all processes in P_j contain only correct processes. Formally $\exists t_0 \in \mathcal{T}, \forall t \geq t_0, \forall p \in P_j \setminus F(t), H(p,t).\mathit{Quorum} \subseteq \mathit{correct}(F)$.

($\Pi\Omega 2$) Eventually *lbound* outputs by all processes in P_j are the same. Formally, $\exists t_0 \in \mathcal{T}, \forall t_1, t_2 \geq t_0, \forall p_1 \in P_j \setminus F(t_1), \forall p_2 \in P_j \setminus F(t_2), H(p_1,t_1).\mathit{lbound} = H(p_2,t_2).\mathit{lbound}$.

($\Pi\Omega 3$) Eventually the *isLeader* outputs on any correct process in P_j do not change. Formally, $\exists t_0 \in \mathcal{T}, \forall t > t_0, \forall p \in P_j \setminus F(t), H(p,t).\mathit{isLeader} = H(p,t_0).\mathit{isLeader}$.

($\Pi\Omega 4$) There is at least one eventual leader. Formally, $|\mathit{Leaders}(P_j)| \geq 1$.

($\Pi\Omega 5$) The number of eventual leaders is eventually bounded by the *lbound* outputs. Formally, $\exists t_0 \in \mathcal{T}, \forall t \geq t_0, |\mathit{Leaders}(P_j)| \leq H(p,t).\mathit{lbound}$.

We call a component that satisfies the liveness properties ($\Pi\Sigma 3$, $\Pi\Omega 2$–5) a *live component*, and other components *non-live components*. A live component must have at least one correct process by ($\Pi\Omega 4$), but a non-live component may not have any correct processes.

We now provide some intuition behind the definition by explaining (a) why intuitively failure detectors in Π_k can help solving k-set agreement; and (b) why Π_k may be strictly weaker than $\Omega''_k \times \Sigma$.

First, from properties ($\Pi\Sigma 1$) and ($\Pi\Sigma 2$), processes in one component can easily isolate themselves from other components by checking if their *Quorum* outputs intersect. Second, if we look at one live component P_j, the properties on this component match the definition of $\Omega''_{k_j} \times \Sigma$ restricted to P_j, with $k_j = \mathit{lbound}(P_j)$. One implication is that if we have only one component, Π_k is reduced to $\Omega''_k \times \Sigma$. A more general implication is that if we are able to run a set agreement algorithm in isolation in component P_j, eventually some correct process will decide and the liveness of k-set agreement is guaranteed. Third, every component P_i satisfies safety properties that match to $\Omega''_{k_i} \times \Sigma$ with $k_i = \mathit{lbound}(P_i)$. Thus if we are able to run a set agreement algorithm in isolation on P_i, it should not have more than k_i decisions. Finally, by property ($\Pi\Omega 1$), we know that if we run the algorithm in parallel on all components, then we have at most k decisions, and thus the safety property of k-set agreement is guaranteed.

Class Π_k is strictly weaker than $\Omega''_k \times \Sigma$ with the following intuitive reasons. First, because of its partition properties ($\Pi\Sigma 1$–2) it does not have enough information to construct a global quorum system required by Σ. Second, those non-live components are free to have more eventual leaders than their *lbound* values. Therefore, Π_k does not have enough information either to construct a failure detector in Ω''_k, which requires that the total number of eventual leaders among all processes be at most k.

One additional remark is that the *lbound* values of an entire component could always be 0, which implies that the component is passive and cannot make any decision by itself.

3.2 Splittable partitioned failure detectors Π^S_k

For a statically partitioned failure detector in Π_k, a partition is always fixed throughout a run. In this section, we relax this restriction and define a class of splittable partitioned failure detectors Π^S_k. Generally, Π^S_k allows a partition to be split further during a run, which both weakens Π_k and makes it more realistic for practical scenarios. We first define *partition tree* and *partition split history*, which represent the partition splitting process, and then provide the full specification of Π^S_k.

A tree is a common data structure, with its associated concepts such as root, nodes, parent nodes, child nodes, ancestors, descendants, leaf nodes, internal nodes, etc. A *partition tree* Γ of processes P is a tree structure that satisfies the following properties: (a) each node of Γ is a non-empty subset of P; (b) root of Γ is P; (c) for any node N and its children $N_1, \ldots, N_s$ with $s \geq 1$, $\{N_1, \ldots, N_s\}$ is a partition of N. Whenever it is clear, we do not distinguish the node of tree Γ and the partitioned component associated with the node. Let $N(\Gamma)$ be the set of nodes and $L(\Gamma)$ be the set of leaf nodes in Γ.

A *partition split history* based on a partition tree Γ is a function $S : P \times \mathcal{T} \rightarrow N(\Gamma)$. Informally, $S(p,t)$ denote the tree node that p belongs to at time t. Formally, function S satisfies (a) for all $p \in P$ and all $t \in \mathcal{T}$, $p \in S(p,t)$; (b) for all $p \in P$ and $t, t' \in \mathcal{T}$ with $t < t'$, either $S(p,t) = S(p,t')$ or $S(p,t)$ is an ancester of $S(p,t')$; (c) for all $p \in P$, there exists a time t such that for all $t' > t$, $S(p,t')$ is a leaf node. Note that we allow p to skip some of the internal nodes in the partition split history. This allows our definition to include static partitions as a special case.

The output of a failure detector in class Π_k^S is a tuple $(\mathit{isLeader}, \mathit{lbound}, \mathit{Quorum}, \mathit{cid})$. Among them, the *isLeader*, *lbound* and *Quorum* outputs have the same value ranges and same informal meanings as the corresponding outputs in Π_k. The new output *cid* is an identifier representing the current component of a process. Instead of using unique numerical IDs for different components in the partition tree, we generalize it as follows so that we can generate node IDs for statically partitioned failure detectors. The values of *cid* are drawn from a node ID set $\mathcal{N}$. Set $\mathcal{N}$ is associated with a relation $\equiv$. Informally, we require that in any run, *cid* together with the $\equiv$ relation can distinguish components in the partition tree of the run. For any component P_j in the partition tree Γ, we define $\mathit{lbound}(P_j)$ and $\mathit{Leaders}(P_j)$ in the same way as in Section 3.1.

A failure detector $\mathcal{D}$ is in the class Π_k^S if for any failure pattern F and any failure detector history $H \in \mathcal{D}(F)$, there exists a partition tree Γ and a partition split history S based on Γ, such that H satisfies the following set of safety and liveness properties. The safety properties are:

($\Pi^S\Sigma 1$) The quorum output of a process p is always contained within p's current component. Formally, $\forall t \in \mathcal{T}, \forall p \notin F(t), H(p,t).\mathit{Quorum} \subseteq S(p,t)$.

($\Pi^S\Sigma 2$) The quorum outputs of two processes intersect if their current components intersect. Formally, $\forall t_1, t_2 \in \mathcal{T}, \forall p_1 \notin F(t_1), \forall p_2 \notin F(t_2), S(p_1,t_1) \cap S(p_2,t_2) \neq \emptyset \Rightarrow H(p_1,t_1).\mathit{Quorum} \cap H(p_2,t_2).\mathit{Quorum} \neq \emptyset$.

($\Pi^S\Omega 1$) The sum (taken among all leaf nodes) of the maximum *lbound* outputs (taken among processes in one leaf node) does not exceed k. Formally, $\sum_{P_i \in L(\Gamma)} \mathit{lbound}(P_i) \leq k$.

($\Pi^S C1$) The *cid* outputs together with the $\equiv$ relation can distinguish different components in the tree Γ. Formally, $\forall t_1, t_2 \in \mathcal{T}, \forall p_1 \notin F(t_1), \forall p_2 \notin F(t_2), H(p_1,t_1).\mathit{cid} \equiv H(p_2,t_2).\mathit{cid} \Leftrightarrow S(p_1,t_1) = S(p_2,t_2)$.

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

($\Pi^S\Sigma 3$) Eventually the quorum outputs by all processes in P_j contain only correct processes. Formally, $\exists t_0 \in \mathcal{T}, \forall t \geq t_0, \forall p \in P_j \setminus F(t), H(p,t).\mathit{Quorum} \subseteq \mathit{correct}(F)$.

($\Pi^S\Omega 2$) Eventually *lbound* outputs by all processes in P_j are the same. Formally, $\exists t_0 \in \mathcal{T}, \forall t_1, t_2 \geq t_0, \forall p_1 \in P_j \setminus F(t_1), \forall p_2 \in P_j \setminus F(t_2), H(p_1,t_1).\mathit{lbound} = H(p_2,t_2).\mathit{lbound}$.

($\Pi^S\Omega 3$) Eventually the *isLeader* outputs on any correct process in P_j do not change. Formally, $\exists t_0 \in \mathcal{T}, \forall t > t_0, \forall p \in P_j \setminus F(t), H(p,t).\mathit{isLeader} = H(p,t_0).\mathit{isLeader}$.

($\Pi^S\Omega 4$) There is at least one eventual leader. Formally, $|\mathit{Leaders}(P_j)| \geq 1$.

($\Pi^S\Omega 5$) The number of eventual leaders is eventually bounded by the *lbound* outputs. Formally, $\exists t_0 \in \mathcal{T}, \forall t \geq t_0, \forall p \in P_j \setminus F(t), |\mathit{Leaders}(P_j)| \leq H(p,t).\mathit{lbound}$.

The leaf node components that satisfy all the liveness properties are called live components. We say that failure detector output $H(p,t)$ *appears in a node* N if $S(p,t) = N$. Note that the liveness properties have exactly the same statement as those for the statically partitioned failure detectors. The safety properties have some differences that are further explained below.

First, for property ($\Pi^S\Sigma 2$), it requires that any *Quorum* output appearing in node N intersect with all *Quorum* outputs appearing in N's ancestors. This property is important to k-set agreement algorithms to ensure a kind of inheritance of decision values from ancestor nodes. In Section 6 we sketch a proposal on how to achieve this property.

Second, for property ($\Pi^S\Omega 1$), we want to emphasize that the maximum *lbound* value for a leaf component P_i is taken among all *lbound* values in the *entire history* of all processes in P_i, not just among the *lbound* values appearing in leaf node P_i. This is another important property to guarantee that the number of possible decisions of a k-set agreement algorithm does not exceed k.

Finally, for property ($\Pi^S C1$), it says that the component ID *cid* contains enough information to distinguish two components, but *cid* may not provide other information about components, such as whether one component is an ancestor of the other component.

It is much less intuitive why Π^S_k still ensures the solvability of k-set agreement. Indeed, the algorithm provided in Section 5 has several subtle points and its proof is much more complicated because of the dynamic splitting of partitions. As for the strength of Π^S_k comparing to Π_k, intuitively Π^S_k is more flexible so it should be strictly weaker.

4 Relationship Lattices of Partitioned Failure Detectors

In this section, we characterize the strength of the partitioned failure detectors. In particular, we put the three families of failure detectors, $\{\Omega''_z \times \Sigma\}_{1\le z\le k}$, $\{\Pi_z\}_{1\le z\le k}$, $\{\Pi^S_z\}_{1\le z\le k}$, into two relationship lattices depicted in Figure 1: one for model $\mathcal{A}$, and the other for models $\mathcal{A}[\Sigma]$ and $\mathcal{A}[Maj]$. We show two types of results: (a) all arrows in the lattice diagrams indeed correspond to possible transformations; and (b) all directed paths not existing in the lattice diagrams indeed correspond to impossible transformations. The results, especially the proofs, which for some theorems are quite technically involved, provide precise explanations and insights on why the lattice structures hold, and they match the intuitions explained in the previous section.

4.1 Possible transformations

All possible transformations in the relationship lattices are relatively easy to show, and are covered by the following theorem.

Theorem 1 *The following results hold in models $\mathcal{A}$, $\mathcal{A}[\Sigma]$ and $\mathcal{A}[Maj]$:*
(1) For all $k \ge 2$, $\Omega''_k \times \Sigma \preceq \Omega''_{k-1} \times \Sigma$, $\Pi_k \preceq \Pi_{k-1}$, and $\Pi^S_k \preceq \Pi^S_{k-1}$.
(2) For all $k \ge 1$, $\Pi_k \preceq \Omega''_k \times \Sigma$.
(3) For all $k \ge 1$, $\Pi^S_k \preceq \Pi_k$.
(4) $\Omega''_1 \times \Sigma$, Π_1 and Π^S_1 are equivalent.

Part (1) is true by mere definition. Part (2) is true because $\Omega''_k \times \Sigma$ can be viewed as Π_k with P as a single component in a static partition. Part (3) is true because when we use the quorum outputs of Π_k as the *cid* outputs in Π^S_k and define relation $\equiv$ as two quorums intersecting, a static partition can be shown as a special case of splittable partition. Finally, for part (4), in Π_1 or Π^S_1, only one component has *lbound* $= 1$ (so it is a live component) and all other components must have *lbound* $= 0$, and thus processes can use the outputs of the only live component as the outputs in $\Omega''_1 \times \Sigma$.

For models $\mathcal{A}[\Sigma]$ and $\mathcal{A}[Maj]$, we need to know the strength of the models. In particular, we would like to know whether Σ can be used to implement Ω''_k, and in what situation it can be done. This is covered by the following theorem, which is an extension to a result in [6] that covers the case of $k = 1$.

Theorem 2 *When $n \le 2k$, Σ can be transformed into Ω''_k, or equivalently, Ω''_k can be implemented in $\mathcal{A}[\Sigma]$ and $\mathcal{A}[Maj]$.*

When $n > 2k$, Σ cannot be transformed into Ω''_k, Π_k or Π^S_k. This is a consequence of Theorem 7 (1).

4.2 Impossible transformations in model $\mathcal{A}$

For the impossible transformations in model $\mathcal{A}$, we need to show that there is no transformation from class $\mathcal{C}_1$ to class $\mathcal{C}_2$ if there is no directed path from $\mathcal{C}_1$ to $\mathcal{C}_2$ in the lattice diagram of Figure 1 (a). Showing all these impossible transformations would be tedious, but fortunately with Theorem 1, all impossible transformations are covered by the following critical impossible transformations: (a) Π_2 cannot be transformed into $\Omega''_k \times \Sigma$, i.e., $\Pi_2 \not\succeq \Omega''_k \times \Sigma$; (b) $\Pi^S_2 \not\succeq \Pi_k$; and (c) $\Omega''_k \times \Sigma \not\succeq \Pi^S_{k-1}$ for $k \geq 2$.

The reason is based on the following important fact: If $\mathcal{C}_1 \not\succeq \mathcal{C}_2$, $\mathcal{C}_1 \succeq \mathcal{C}_3$, and $\mathcal{C}_4 \succeq \mathcal{C}_2$, then $\mathcal{C}_3 \not\succeq \mathcal{C}_4$. So, for example if we know $\Pi_2 \not\succeq \Omega''_k \times \Sigma$, then we know that for any Π_y and Π^S_y with $y \geq 2$ and any $\Omega''_z \times \Sigma$ with $z \leq k$, we have $\Pi_y \not\succeq \Omega''_z \times \Sigma$ and $\Pi^S_y \not\succeq \Omega''_z \times \Sigma$, because $\Pi_2 \succeq \Pi_y \succeq \Pi^S_y$ and $\Omega''_z \times \Sigma \succeq \Omega''_k \times \Sigma$. Our results in this section focus on these critical impossible transformations.

We first study the relation between Π_2 and $\Omega''_k \times \Sigma$. The following theorem shows that Π_2 cannot be transformed into Σ.

Theorem 3 *Π_2 cannot be transformed into Σ in model $\mathcal{A}$.*

The proof of this theorem is relatively easy. The main idea is that, assuming the existence of a transformation from Π_2 to Σ, we can have two partitioned components by the definition of Π_2, and we isolate the two components long enough to force processes in each component to generate *Quorum* outputs of Σ that are contained within each component. This contradicts the property of Σ that requires any two *Quorum* outputs intersect. It reflects the intuition that partitioning (even with only two components) weakens the global quorum requirement.

Theorem 3 is enough to show that Π_2 cannot be transformed into $\Omega''_k \times \Sigma$. However, to understand the strength of Π_k, we are still interested in knowing if Π_k (or Π^S_k) can be transformed into $\Omega''_{k'}$ in model $\mathcal{A}$. The following theorem completely characterizes the transformability from Π_k or Π^S_k to $\Omega''_{k'}$ in model $\mathcal{A}$.

Theorem 4 *In model $\mathcal{A}$, for any $k \geq 2$, Π_k (or Π^S_k) can be transformed into $\Omega''_{k'}$ if and only if $k' \geq k$ and $n \leq 2k' - k + 1$.*

This is an important theorem showing that partitioning not only weakens the global quorum requirement of Σ, but in most cases it also weakens the leader election part of Ω''_k. To prove the only-in part, we construct a failure detector in Π_k with k partitioned components, such that $k-1$ of them are singleton and live components while the only big component is non-live. If a transformation exists when $k' < k$ or $n > 2k' - k + 1$, by isolating each component, we can force the $k-1$ singleton components to generate $k-1$ eventual leaders in $\Omega''_{k'}$, then we can manipulate the big non-live component to force it to generate at least $k' - k + 2$ eventual leaders, which violates the requirement of at most k' eventual leaders of $\Omega''_{k'}$. The proof of the if part uses the transformation technique in Theorem 2.

We now compare the strength of splittable partitioned failure detectors to the statically partitioned failure detectors. The following theorem shows the critical impossible transformation from Π^S_2 to Π_k, which indicates that even one dynamic splitting is enough to weaken the failure detectors.

Theorem 5 *Π^S_2 cannot be transformed into Π_k in model $\mathcal{A}$.*

The idea of the proof is that, assuming the existence of a transformation algorithm T, for the failure detector in Π^S_2, we do not partition first but let T run and check the *Quorum* outputs of Π_k to see how it partitions. If there is a component with multiple processes, then in Π^S_2 we split that component into two and isolate them so that their *Quorum* outputs of Π_k do not intersect, which contradicts to ($\Pi\Sigma 2$) of Π_k. For the other case when all components of Π_k contain only one process, we can suppress each live component one by one to force the outputs of Π_k to have at least $k+1$ components with *lbound* ≥ 1, which contradicts to ($\Pi\Omega 1$).

We now compare the strength of $\Omega''_k \times \Sigma$, Π_k, and Π^S_k to $\Omega''_{k-1} \times \Sigma$, Π_{k-1}, and Π^S_{k-1}. The following theorem compares their strength by their ability of solving $(k-1)$-set agreement, based on a result in [14], which shows that in model $\mathcal{A}[\mathit{Maj}]$ with $n > 2k$, if class Ω_z can be used to solve k-set agreement, then $z \leq k$.

Theorem 6 *For any $k \geq 2$ and any $n \geq 2k-1$, failure detector classes $\Omega''_k \times \Sigma$, Π_k, and Π^S_k cannot be used to solve $(k-1)$-set agreement in model $\mathcal{A}$. Together with Theorem 11, this implies that classes $\Omega''_k \times \Sigma$, Π_k, and Π^S_k cannot be transformed into classes $\Omega''_{k-1} \times \Sigma$, Π_{k-1}, and Π^S_{k-1}.*

Note that the condition $n \geq 2k-1$ is tight, because when $n \leq 2k-2$, by Theorem 2 Σ itself can implement Ω''_{k-1} and thus solves $(k-1)$-set agreement. We also have direct proofs to show the strength of $\Omega''_k \times \Sigma$, Π_k, and Π^S_k comparing with $\Omega''_{k-1} \times \Sigma$, Π_{k-1}, and Π^S_{k-1}. In particular, we have new results showing that $n \geq 2k-1$ is not needed to show that Π_k and Π^S_k cannot be transformed into Π_{k-1} and Π^S_{k-1}.

Theorem 7 *In model $\mathcal{A}$, when $k \geq 2$, we have*
(1) for any $n \geq 2k-1$, $\Omega''_k \times \Sigma$ cannot be transformed into Π^S_{k-1}; and
(2) for any $n > k$, Π_k cannot be transformed into Π^S_{k-1}.

The proof of (1) has an important technique used in later proofs such as that of Theorem 10.

By now, we have obtained a complete characterization of the relationship lattice for the partitioned failure detector in model $\mathcal{A}$, as shown in Figure 1 (a). From the theorems and their proofs, we understand that static partitions (even with just two components) weaken both Σ and Ω''_k (in most cases), and dynamic splitting of partitions (even once) further weakens failure detectors.

4.3 Impossible transformations in models $\mathcal{A}[\Sigma]$ and $\mathcal{A}[\mathit{Maj}]$

We now study the relationship of partitioned failure detectors in models $\mathcal{A}[\Sigma]$ and $\mathcal{A}[\mathit{Maj}]$. As explained in the introduction, we would like to see if adding Σ or the majority requirement would reverse the effect of partitioning and collapse the relationship lattice. Our results in this section show that it is not the case.

By Theorem 2, we know that in model $\mathcal{A}[\Sigma]$ and $\mathcal{A}[\mathit{Maj}]$, Ω''_k can be implemented if $n \leq 2k$. So, we only investigate the cases when $n > 2k$. The following theorem compares Ω''_k, Π_k, and Π^S_k with Ω''_{k-1}, Π_{k-1}, and Π^S_{k-1}, and it is a direct consequence of Theorem 6 and 7 (1).

Theorem 8 *When $n \geq 2k-1$, Ω''_k, Π_k, and Π^S_k cannot solve $(k-1)$-set agreement and cannot be transformed into Ω''_{k-1}, Π_{k-1}, and Π^S_{k-1} in models $\mathcal{A}[\Sigma]$ and $\mathcal{A}[\mathit{Maj}]$.*

We now compare the strengths of families $\{\Omega''_z\}_{2\leq z\leq k}$, $\{\Pi_z\}_{2\leq z\leq k}$, and $\{\Pi^S_z\}_{2\leq z\leq k}$.

Theorem 9 *When $n \geq 4k+2$, Π_2 cannot be transformed into Ω''_k in model $\mathcal{A}[\Sigma]$ or $\mathcal{A}[\mathit{Maj}]$.*

The basic idea to prove this theorem is again manipulating the non-live component allowed by Π_2 to cause the transformation algorithm generating more leaders than required by Ω''_k. The complication comes when we need to comply with the model $\mathcal{A}[\Sigma]$ or $\mathcal{A}[\mathit{Maj}]$. This is where the condition $n \geq 4k+2$ is used to allow both the construction of a global majority quorum and our manipulation to generate extra leaders.

Theorem 10 *When $n \geq 4k+2$, Π^S_2 cannot be transformed into Π_k in model $\mathcal{A}[\Sigma]$ or $\mathcal{A}[\mathit{Maj}]$.*

The proof of this theorem is the most complicated among the proofs in this section. It combines the techniques used in the proofs of several theorems, in particular Theorems 5, 7 (1), and 9.

With the last two theorems, we now have a complete characterization of the relationship lattice for partitioned failure detectors in models $\mathcal{A}[\Sigma]$ and $\mathcal{A}[\mathit{Maj}]$ when $n \geq 4k+2$, as shown in Figure 1 (b). One open problem left is what happens when $2k+1 \leq n \leq 4k+1$. This could be the limited case when Σ does collapse the relationship lattice.

```
On proposer $p$ with unique id $i \in \{1, \ldots, n\}$:

Proposer input and state variables:
1  $proposal$: the initial proposal value, read-only
2  ($isLeader, lbound, Quorum, cid$): output of $\Pi_k^S$,
       read-only
3  $p\_round$: current round number, initially $i$
4  $p\_Rounds$: top $n$ rounds that $p$ sees, initially $\{i\}$
5  $p\_cid$: current $cid$, initially 0
6  $taskid$: unique id for each task, initially 0
7  $incflag$: flag to increase $p\_round$, initially $False$

Run periodically if not decided yet
8  **if** $isLeader$ **and** no task 1 running **then**
9    $taskid \leftarrow taskid + 1$
10   **if** $p\_round \notin top(p\_Rounds, lbound)$
       **or** $incflag$ **then**
11     $p\_round \leftarrow p\_round + t \cdot n$ such that
         $p\_round + t \cdot n > \max p\_Rounds$
12     $p\_Rounds \leftarrow p\_Rounds \cup_n \{p\_round\}$
13   $incflag \leftarrow False$; $p\_cid \leftarrow cid$
14   start task 1
```

Figure 2: Part I of k-set agreement algorithm using Π_k^S: proposer thread, task driver.

```
Task 1: a new round of $p$
15 send (PREPARE, $p\_round, p\_Rounds, lbound,$
     $p\_cid, taskid$) to all acceptors
16 **repeat periodically**
17   $p\_Quorum \leftarrow Quorum$
18 **until** $p\_Quorum \subseteq \{q |$ received ACK-PREP or
     NACK-PREP message with $taskid$ from $q\}$
     **or** $p\_cid \not\equiv cid$
19 $M_1 \leftarrow \{$(ACK-PREP, $R, TS, v, around, taskid$)
     received from acceptors in $p\_Quorum\}$
20 $M_2 \leftarrow \{$(NACK-PREP, $R, taskid$) received
     from acceptors in $p\_Quorum\}$
21 $p\_Rounds \leftarrow p\_Rounds \cup_n (\bigcup_{m \in M_1 \cup M_2} m.R)$
22 **if** (1) $M_2 \neq \emptyset$ **or** (2) some received $R$'s in $M_1$
     are different **or** (3) $p\_cid \not\equiv cid$ **then**
23   stop this task
24 **if** $\forall m \in M_1, m.v = \perp$ **then** $est \leftarrow proposal$
25 **else** $est \leftarrow m.v$ with $m \in M_1$ such that $m.TS$ is
     the highest based on $\preceq_n$, and $m.around$ is the
     highest among $m' \in M_1$ with $m'.TS = m.TS$
26 send (ACCEPT, $est, p\_round, p\_Rounds, p\_cid,$
     $taskid$) to all acceptors
27 **repeat periodically**
28   $p\_Quorum \leftarrow Quorum$
29 **until** $p\_Quorum \subseteq \{q |$ received ACK-ACC or
     NACK-ACC message with $taskid$ from $q\}$
     **or** $p\_cid \not\equiv cid$
30 **if** $p\_cid \not\equiv cid$ **then** $incflag \leftarrow True$; stop this task
31 **else if** received (NACK-ACC, $R, taskid$) from an
     acceptor in $p\_Quorum$ **then**
32   **if** $R = \perp$ **then** $incflag \leftarrow True$
33   **else** $p\_Rounds \leftarrow p\_Rounds \cup_n R$
34   stop this task
35 decide($est$)
```

Figure 3: Part II of k-set agreement algorithm using Π_k^S: proposer thread, task for one round.

5 Algorithm for k-set agreement using Π_k^S

In [5] we already presented an extension of the Paxos algorithm [11] to solve k-set agreement in model $\mathcal{A}[Maj]$ with failure detectors in Ω''_k. The algorithm presented in Figures 2–4 is a further extension to use failure detectors in Π_k^S. We use similar terminologies as in the Paxos algorithm summarized in [12]. Each process behaves both as a proposer and an acceptor.[4] As with the Paxos algorithm, each proposer's round has two phases. In the first phase, the *preparation phase* (lines 15–25), the proposer p sends a PREPARE message to all acceptors, and collects the responses from a quorum of acceptors. Based on the responses, it selects a new estimate *est* to the decision value. In the second phase, the *acceptance phase* (lines 26–35), p sends an ACCEPT message with the new estimate *est* to all acceptors. If p receives ACK-ACC responses from a quorum of acceptors, then it decides on *est*; otherwise it will start a new round.

To solve k-set agreement, we first use the extension in [5] to allow an acceptor q to accept multiple rounds instead of a single round. To support this, we first define $top(R, m)$, $\cup_m$, and $\preceq_m$, where $top(R, m)$ is a function returning the m highest round numbers in R, $\cup_m$ is an operator such

[4] We can also allow proposers and acceptors to be separate entities, but in this case we need to adjust our failure detector definition such that the leader-related properties refer to proposers while the quorum-related properties refer to acceptors.

```
On acceptor $q$ (an acceptor is also a proposer):

Acceptor input and state variables:
36  $a\_cid$: $cid$ output of $\Pi_k^S$, read-only
37  $a\_Rounds$: top $n$ rounds that $q$ sees, initially $\emptyset$
38  $a\_est$: estimate of the final value, initially $\bot$
39  $a\_TS$: top $n$ rounds that $q$ sees when $q$ accepts a
        value, initially $\emptyset$
40  $a\_round$: estimate of the final round number,
        initially 0

41  Upon receipt of $(\text{PREPARE}, r, R, lb, pcid, taskid)$
        from $p$
42    if $a\_cid \not\equiv pcid$ then
        send $(\text{NACK-PREP}, \emptyset, taskid)$ to $p$
43    else
44      $a\_Rounds \leftarrow a\_Rounds \cup_n R$
45      if $r \notin top(a\_Rounds, lb)$ then send
          $(\text{NACK-PREP}, a\_Rounds, taskid)$ to $p$
46      else send $(\text{ACK-PREP}, a\_Rounds, a\_TS,$
          $a\_est, a\_round, taskid)$ to $p$

47  Upon receipt of
        $(\text{ACCEPT}, v, pround, R, pcid, taskid)$ from $p$
48    if $a\_cid \not\equiv pcid$ then
        send $(\text{NACK-ACC}, \bot, taskid)$ to $p$
49    else
50      $a\_Rounds \leftarrow a\_Rounds \cup_n R$
51      if $R \neq a\_Rounds$ then send
          $(\text{NACK-ACC}, a\_Rounds, taskid)$ to $p$
52      else
53        $(a\_est, a\_TS, a\_round) \leftarrow (v, R, pround)$
54        send $(\text{ACK-ACC}, taskid)$ to $p$
```

Figure 4: Part III of k-set agreement algorithm using Π_k^S: acceptor thread.

that $R_1 \cup_m R_2 = top(R_1 \cup R_2, m)$, and $\preceq_m$ is a partial order such that $R_1 \preceq_m R_2$ if and only if $R_1 \cup_m R_2 = R_2$. We then have proposers and acceptors maintain variables *p_Rounds* and *a_Rounds* respectively, which is a set of at most n round numbers. They exchange their *p_Rounds* and *a_Rounds* values and merge them using $\cup_n$ so that their values eventually converge (lines 21, 33, 44, and 50).

One key point is that acceptor q only accepts rounds in $top(a\_Rounds, lbound)$, where *lbound* is the failure detector output of a proposer (line 45). The second key point is that proposer p only selects a new value v as its *est* value with the highest *TS* value based on the order $\preceq_n$ (line 25), where *TS* is the *a_Rounds* value when an acceptor accepts the value v (line 53). We can show that $\preceq_n$ among these *TS* is a total order because of the quorum intersection guarantee, and thus p can always select a v with the highest *TS* in line 25. In [5], these two points are enough to guarantee the Uniform k-Agreement property (deciding at most k values) in model $\mathcal{A}[Maj]$ with Ω''_k. For liveness, eventually only the leaders may start a new round (line 8), and they make sure that the new round number *p_round* is in $top(p\_Rounds, lbound)$ (lines 10–12). More details are covered in [5].

To use failure detectors in Π_k^S, we further include the following additions. First, we use *Quorum* outputs of Π_k^S as the stopping condition of each phase of a proposer instead of the majority condition (lines 16–18, 27–29). Second, we use the *cid* output of Π_k^S to ensure that proposer p and its related acceptors always work in the same partitioned component during one round of the proposer (lines 22, 30, 42, and 48). If the proposer or an acceptor moves to a new component (because of partition splitting), the proposer stops its current round. This leads to a subtle scenario when a proposer may enter two acceptance phases with the same (p_round, p_Rounds) pair because of its *cid* changes. This scenario may lead to more than k decision values, so it should be avoided. We use a Boolean variable *incflag* to guarantee that whenever an acceptance phase is stopped because of a *cid* change, the next acceptance phase must have a higher *p_round* value. Third, in line 25 when proposer p selects a new *est* value, it needs to make sure that among the highest *TS* values, it selects one with the highest $around$ value. This is to avoid another subtle scenario that might lead to the violation of the Uniform k-Agreement property.

The following theorem summarizes the correctness of the algorithm with a complete proof in the appendix.

Theorem 11 *The algorithm in Figures 2, 3 and 4 solves the k-set agreement problem with any failure detector in Π_k^S.*

The proof of the Uniform k-Agreement property is complicated due to the dynamic splitting of the failure detectors in Π_k^S. The key idea is as follows. We consider *successful acceptance phases*, those acceptance phases in which the proposer decides. Due to the checking of *cid*, each successful acceptance phase runs in a particular node of the partition tree. We define *critical nodes* to be those nodes that have successful acceptance phases but their ancestors do not have any successful acceptance phases. By definition, critical nodes do not intersect with one another. Since every decision value occurs in some successful acceptance phase, which runs in some node, we can say that every decision value occurs in a subtree rooted at some critical node. The key proof is to show that for any critical node P_j, there are at most *lbound*(P_j) decision values occur in the subtree rooted at P_j. This proof is by tracing back each decision value in a similar way as the proof of non-partitioned algorithm in [5], but is more complicated due to the dynamic partitioning. Once this result is established, it is easy to apply property $(\Pi^S\Omega 1)$ to show that there are at most k decision values in the run.

6 Future Work

There are several directions we plan to take for our future work. First, we will look into the implementation of partitioned failure detectors in weak system environments such as partitionable systems. This will help us understand the weak conditions that the k-set agreement requires. One difficult property to implement is $(\Pi^S\Sigma 2)$, which requires that any quorum intersect with all quorums that appear in its ancestor nodes. For this property, we have the following proposal as an example. Suppose $n = m^2$ processes are arranged (virtually) into an $m \times m$ grid. A quorum system can be designed such that the union of any one row and any one column is a quorum. Then if the split only occurs at the row boundaries, we can see that the row section of a quorum in a component after a split always intersects with the column sections of all quorums of all its ancestors.

Second, there are opportunities to further weaken the failure detectors. For example, property $(\Pi^S\Sigma 2)$ may be weakened, and *cid* outputs may not be necessary. Another possibility is to allow component merges on top of splitting, but it could be much more complicated.

The partition approach is a general one that we have shown in this paper and in [4] to be effective in weakening a number of failure detectors. One may also look into defining it as a general operator on failure detectors and study the properties of partition operators.

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

[4] W. Chen, Y. Chen, and J. Zhang. On failure detectors weaker than ever. Technical Report MSR-TR-2007-50, Microsoft Research, May 2007.

[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] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. Shared memory vs. message passing. Technical Report IC/2003/77, EPFL, Dec. 2003.

[7] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, and S. Toueg. The weakest failure detectors to

solve certain fundamental problems in distributed computing. In *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing*, pages 338–346, July 2004.

[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] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[12] L. Lamport. Paxos made simple. *ACM SIGACT News*, 32(4):51–58, 2001.

[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] G. Neiger. Failure detectors and the wait-free hierarchy. In *Proceedings of the 14th ACM Symposium on Principles of Distributed Computing*, pages 100–109, Aug. 1995.

[17] 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.

[18] 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.

[19] 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 Proofs of Theorems related to the Relationship Lattices

Theorem 1 *The following results hold in models $\mathcal{A}$, $\mathcal{A}[\Sigma]$ and $\mathcal{A}[Maj]$:*
(1) For all $k \geq 2$, $\Omega''_k \times \Sigma \preceq \Omega''_{k-1} \times \Sigma$, $\Pi_k \preceq \Pi_{k-1}$, and $\Pi^S_k \preceq \Pi^S_{k-1}$.
(2) For all $k \geq 1$, $\Pi_k \preceq \Omega''_k \times \Sigma$.
(3) For all $k \geq 1$, $\Pi^S_k \preceq \Pi_k$.
(4) $\Omega''_1 \times \Sigma$, Π_1 and Π^S_1 are equivalent.

Proof. Obviously showing the results in model $\mathcal{A}$ is enough. Part (1) holds directly by the definitions of the failure detector classes. Part (2) holds because a failure detector in $\Omega''_k \times \Sigma$ can be viewed as a failure detector in Π_k with the full process set P as the single component in the static partition.

For Part (3), we need to construct a transformation from Π_k to Π^S_k. We use the same output of Π_k as the output of Π^S_k, and the only additional output to take care of is *cid* in Π^S_k. For this output, we use the *Quorum* output values of Π_k as the *cid* output values. This means the range of *cid* is 2^P, the set of all subsets of P, and the $\equiv$ relation is defined as $A \equiv B$ if and only if $A \cap B \neq \emptyset$. For each run of the algorithm T with a failure detector in Π_k, there is a static partition $\{P_1, \ldots, P_s\}$ by the definition of Π_k. We can define a partition tree and a partition split history to match this static partition: a partition tree Γ has the root P with its s children $P_1, \ldots, P_s$ as the leaf nodes, and the partition split history S is such that $S(p,t) = P_j$ for all $t \in \mathcal{T}$, all $p \in P_j$, and all $j = 1, \ldots, s$. With the above construction, it is straightforward to verify that algorithm T generates outputs of a failure detector that satisfy all properties of Π^S_k. Therefore, $\Pi^S_k \preceq \Pi_k$ for all $k \geq 1$.

Finally we prove Part (4). From (2) and (3), we only need to show that Π^S_1 can be transformed into $\Omega''_k \times \Sigma$ by a transformation algorithm T. Let $(\mathit{isLeader}, \mathit{lbound}, \mathit{Quorum}, \mathit{cid})$ denote the output of a failure detector $\mathcal{D}$ in Π^S_1, and $(\mathit{isLeader}', \mathit{lbound}', \mathit{Quorum}')$ denote the output of a failure detector $\mathcal{D}'$ in $\Omega''_k \times \Sigma$ generated by algorithm T. By the definition of Π^S_1, we know that in any run R, there exists a leaf component P_j such that eventually all correct processes in P_j have $\mathit{lbound} = 1$, exactly one correct process $p_j \in P_j$ is an eventual leader, all other processes $q \notin P_j$ always have $\mathit{lbound} = 0$, any two *Quorum* outputs by processes in P_j intersect, and eventually *Quorum* outputs of processes in P_j contain only correct processes.

Algorithm T works as follows. For the lbound' output of $\mathcal{D}'$, T always sets it to 1, which satisfies properties $(\Omega''1)$ and $(\Omega''2)$ of Ω''_1 (see page 3 for property definitions). For the $\mathit{isLeader}'$ output of $\mathcal{D}'$ on a process p, T sets it to *False* if the *lbound* output on p is 0, otherwise T sets it to *isLeader*. So all processes not in P_j always have $\mathit{isLeader}' = \mathit{False}$. Since eventually *lbound* values and *isLeader* values of processes in P_j do not change, we know that eventually $\mathit{isLeader}'$ outputs will not change, so $(\Omega''3)$ is satisfied. Since no process outside P_j could be an eventual leader (w.r.t. $\mathit{isLeader}'$), eventually $\mathit{isLeader}'$ outputs of processes in P_j are the same as *isLeader* outputs. We know that there is exactly one eventual leader (w.r.t. $\mathit{isLeader}'$), so $(\Omega''4)$ and $(\Omega''5)$ of Ω''_1 hold. Therefore, T generates a failure detector in Ω''_k.

Finally, we need to generate Quorum' outputs of $\mathcal{D}'$ such that they satisfy the requirements of Σ. Initially, Quorum' is set to the full set P. On a process p, if its *lbound* output is 1, then p sends its current *Quorum* output to all processes; if its *lbound* output is 0, then it does not send any message. When a process receives a message with quorum Q in it, it sets its Quorum' output to Q. This transformation guarantees that only processes in P_j can broadcast their quorum outputs. Since quorum outputs in P_j always intersect, all Quorum' outputs intersect. Since the quorum output in P_j eventually contains only correct processes, the Quorum' outputs eventually also contain only correct processes. Therefore, the Quorum' outputs of $\mathcal{D}'$ satisfy the requirements of Σ. □

Theorem 2 *When $n \leq 2k$, Σ can be transformed into Ω''_k, or equivalently, Ω''_k can be implemented in $\mathcal{A}[\Sigma]$ and $\mathcal{A}[Maj]$.*

Proof. We choose a total order "$\leq$" among the subsets of P such that $|A| \leq |B|$ implies $A \leq B$. We design the transformation algorithm T from Σ to Ω_k as follows. Each process p maintains a local variable *Leaders*, which contains a subset of processes. Process p periodically sets its *lbound* output to $|$*Leaders*$|$ and *isLeader* output to *True* if and only if $p \in$ *Leaders*.[5]

Initially p sets *Leaders* to be $\max\{A \subseteq P \mid |A| \leq k\}$ according to the total order $\leq$. Process p periodically checks if its *Quorum* output satisfies *Quorum* $\leq$ *Leaders*; if so, p sets its *Leaders* to the *Quorum* output. Process p also periodically sends out its *Leaders* value to all processes. When p receives a *Leaders*$_q$ message from a process q, if *Leaders*$_q$ $\leq$ *Leaders*, then p sets its *Leaders* to *Leaders*$_q$.

We now show that algorithm T is a correct transformation. First, the *Leaders* output on a process p can only decrease according to the total order $\leq$, so we know that the size of *Leaders* is always at most k, and thus $(\Omega''1)$ holds. Second, there are only a finite number of subsets of P, so eventually on a process p its *Leaders* output stops changing. Moreover, processes exchange their *Leaders* output and if they see a message contains a smaller *Leaders* set, they set it to their own *Leaders* output. Thus, eventually all processes have the same *Leaders* output. So eventually *isLeader* outputs do not change $(\Omega''3)$, *lbound* outputs do not change and are the same $(\Omega''2)$, and the number of eventual leaders is at most the eventual *lbound* value $(\Omega''5)$.

Let L be the final *Leaders* output all processes converge to. To show $(\Omega''4)$, it is enough to show that L has at least one correct process, since that correct process will eventually set its *isLeader* to *True*. There are two cases to consider. In the first case, L is not the initial value of *Leaders*. This means that L equals to a *Quorum* output. By the definition of Σ, it is easy to see that any *Quorum* output contains at least one correct process. Therefore L has at least one correct process. In the second case, L is the initial value of *Leaders*, thus $L = \max\{A \subseteq P \mid |A| \leq k\}$. This means that the query results of *Quorum* on all processes are $\geq L$. If there is a query output of *Quorum* with size k, it must be *Quorum* $= L$. Using the same argument in the first case, L must contain at least one correct process. Otherwise all *Quorum* outputs processes get have sizes larger than k. By the property of Σ, we know that eventually the *Quorum* output contains only one correct process. Therefore, we know that there are at least $k+1$ correct processes in the run. Since $|L| = k$ and $n \leq 2k$, we know that L intersects with the set of correct processes. Hence, the transformation algorithm T is correct. $\square$

Theorem 3 *Π_2 cannot be transformed into Σ in model $\mathcal{A}$.*

Proof. Suppose, for a contradiction, that $\Pi_2 \succeq \Sigma$. Let T be the transformation algorithm that transforms any failure detector $\mathcal{D}$ in Π_2 into a failure detector $\mathcal{D}'$ in Σ. Let (*isLeader*, *lbound*, *Quorum*) be the output of of $\mathcal{D} \in \Pi_2$ and *Quorum*$'$ be the output of $\mathcal{D}' \in \Sigma$. We consider a static partition $\{P_1, P_2\}$ of P. Let p_i be a process in P_i for $i = 1, 2$.

We construct two runs R_1 and R_2 of T with failure detector $\mathcal{D} \in \Pi_2$ as follows. In run R_i, all processes in $P \setminus P_i$ crash at the beginning of the run and all processes in P_i are correct. The outputs of $\mathcal{D}$ on a process $p \in P_i$ in run R_i are set as follows: (a) if $p = p_i$ then *isLeader* is always *True*, otherwise, *isLeader* is always *False*; (b) *lbound* is always 1; and (c) *Quorum* is always P_i. It is easy to verify that such outputs satisfy the definition of Π_2. Therefore, algorithm T generates a failure detector history H_i of $\mathcal{D}'$ that satisfies the requirement of Σ. By the definition of Σ, there is a time t_i at which the *Quorum*$'$ output of $\mathcal{D}'$ on p_i only contains correct processes, that is, $H_i(p_i, t_i) \subseteq P_i$.

Let $t = \max(t_1, t_2)$. We now construct a new run R, in which all processes are correct, and the output of $\mathcal{D}$ on a process $p \in P_i$ is exactly the same as the output of $\mathcal{D}$ on p in run R_i at all times. Again it is easy to verify that all properties of Π_2 are still satisifed. Therefore, algorithm T in this run generates a failure detector history H of $\mathcal{D}'$ that satisfies the requirement of Σ. In run R all messages within the same component P_i sent before t are delivered

[5]Actually, *Leaders* outputs form a failure detector in Ω_k [16, 14], and the way we set *isLeader* and *lbound* from *Leaders* is the transformation from Ω_k to Ω''_k in [5].

exactly at the same time as in run R_i, and all messages across different components are delayed until after time t. Therefore, all processes in P_i behave exactly the same as in run R_i by time t. Thus, at time $t_i \leq t$, T generates a *Quorum*$'$ output value $Q_i \subseteq P_i$ at p_i . This means in run R we find two *Quorum*$'$ outputs Q_1 and Q_2 that do not intersect with each other. This contradicts to property $(\Sigma 1)$ of Σ failure detectors. □

Theorem 4 *In model $\mathcal{A}$, for any $k \geq 2$, Π_k (or Π_k^S) can be transformed into $\Omega''_{k'}$ if and only if $k' \geq k$ and $n \leq 2k' - k + 1$.*

We divide this theorem into the next two lemmas and prove them separately.

Lemma 1 *In model $\mathcal{A}$, for any $k \geq 2$, Π_k cannot be transformed into $\Omega''_{k'}$ if $k > k'$ or $n \geq 2k' - k + 2$.*

Proof. Suppose, for a contradiction, that we have an algorithm T that transforms any failure detector $\mathcal{D}$ in Π_k to a failure detector $\mathcal{D}'$ in $\Omega''_{k'}$. Let (*isLeader*, *lbound*, *Quorum*) denote the output of $\mathcal{D}$, and (*isLeader*$'$, *lbound*$'$) denote the output of $\mathcal{D}'$ generated by algorithm T.

We construct a partition of P, $\{P_0, P_1, \ldots, P_{k-1}\}$, such that P_i contains exactly one process p_i for all $i = 1, \ldots, k-1$, and P_0 contains the rest $n - k + 1$ processes. Since $k \geq 2$, we have at least one component in $P_1, \ldots, P_{k-1}$. We consider a failure pattern in which all processes are correct. We construct a failure detector history H of $\mathcal{D}$ such that (a) *lbound* outputs of all processes are always 1; (b) for $p_i \in P_i$ with $i = 1, \ldots, k-1$, p_i's *isLeader* outputs are always *True*, and p_i's *Quorum* outputs are always $\{p_i\}$; and (c) for $q \in P_0$, if $k > k'$, then always set *Quorum* outputs to P_0; otherwise ($k \leq k'$ and $n \geq 2k' - k + 2$), set the *Quorum* outputs of q such that they always contain $n - k'$ processes in P_0. Therefore, all components $P_1, \ldots, P_{k-1}$ are live components. For component P_0, any two *Quorum* outputs intersect by item (c). This is obvious if $k > k'$. When $k \leq k'$ and $n \geq 2k' - k + 2$, this is because $|P_0| = n - k + 1$, and for any two *Quorum* outputs Q_1 and Q_2, we have $|Q_1| + |Q_2| - |P_0| = 2(n - k') - (n - k + 1) = n - (2k' - k + 1) \geq 1$. Thus failure detector outputs in component P_0 satisfy the safety properties of Π_k. So all failure detector histories conforming with the conditions above satisfy all properties of Π_k. Therefore, we can choose to set *isLeader* and *Quorum* outputs for component P_0 in order to reach a contradiction. The exact way of setting these outputs are given shortly.

We use history H to construct a run R of T. Our objective is to force all p_i's with $i = 1, \ldots, k-1$ to be eventual leaders (w.r.t. *isLeader*$'$ outputs).

To achieve the objective, we construct R as follows. We take round-robin turns among processes $p_1, \ldots, p_{k-1}$. When in turn for p_i, we force the algorithm T to output *isLeader*$'$ $=$ *True* at least once on p_i. To do so, we delay all messages sent to p_i. It is easy to argue that it is indistinguishable on p_i from a run in which all other processes crash at the beginning of p_i's turn. In the latter case, component P_i is the only one that has a correct process. By the properties of Ω''_k, the only process p_i in P_i must eventually output *isLeader*$'$ $=$ *True*. After p_i output *isLeader*$'$ $=$ *True* once, we can deliver all delayed messages to p_i in R, and then start the next turn for the next process in the round-robin order. By continuously doing the above among all processes in $\{p_1, \ldots, p_{k-1}\}$, we construct run R in which all processes in $\{p_1, \ldots, p_{k-1}\}$ output *isLeader*$'$ $=$ *True* infinitely often. Since algorithm T correctly generates *isLeader*$'$ outputs, by property $(\Omega''3)$ of Ω''_k, *isLeader*$'$ outputs do not change on every process eventually. Therefore, we know that all processes $\{p_1, \ldots, p_{k-1}\}$ must be eventual leaders (w.r.t. *isLeader*$'$ outputs) in run R.

Now if $k' < k - 1$, we already have a contradiction because $\Omega''_{k'}$ can only have at most k' eventual leaders. Suppose $k' = k - 1$. We can select one process in P_0 and set its *isLeader* to *True*, and set *isLeader* of all other processes in P_0 to *False*, and we delay all messages sent from $P \setminus P_0$ to P_0. For processes in P_0, it cannot distinguish this run with another run in which all processes in $P \setminus P_0$ have crashed, and P_0 is the only live component, so eventually some process in P_0 sets *isLeader*$'$ to *True*. After this occurs, we can deliver the delayed messages. This procedure can be repeated infinitely often, so there must be at least one process in P_0 where *isLeader*$'$ $=$ *True* infinitely often. That is, there is at least one eventual leader in P_0 (w.r.t. *isLeader*$'$ outputs). Then totally we have at least $k > k'$ eventual leaders (w.r.t. *isLeader*$'$ outputs), contradicting the specification of $\Omega''_{k'}$. Therefore, we have proven the case of $k > k'$.

Next, suppose that $k \leq k'$ and $n \geq 2k' - k + 2$. We further manipulate the run R such that there have to be at least $k' - k + 2$ eventual leaders (w.r.t. *isLeader*$'$ outputs) among processes in P_0.

Let $S_1, S_2, \ldots, S_m$ be all possible subsets of P_0 with exactly $k' - k + 1$ leaders. We take another round-robin turn among S_1 to S_m. When in turn for S_i, we set the *Quorum* outputs of all processes in P_0 to $P_0 \setminus S_i$, and set *isLeader* outputs on one process in $P_0 \setminus S_i$ to *True*, and *isLeader* outputs on all other processes to *False*. Note that $|P_0 \setminus S_i| = n - k + 1 - (k' - k + 1) = n - k'$, so it satisfies condition (c) specified earlier. Also please be reminded that we can set *Quorum* and *isLeader* outputs in P_0 as we wish since it is a non-live component in run R. We delay all messages sent to $P_0 \setminus S_i$. For processes in $P_0 \setminus S_i$, they cannot distinguish this run with another run in which all other processes have crashed at the beginning of S_i's turn. For the latter run, $P_0 \setminus S_i$ is the only live component, so eventually the *isLeader*$'$ output of some process in $P_0 \setminus S_i$ must be set to *True* by the transformation algorithm. Once this occurs, we can deliver all delayed messages and start the next turn.

Note that we can interleave the above round-robin turns with the earlier round-robin turns based on processes $p_1, \ldots, p_k$, so the overall run R can still be constructed.

Now in run R, we argue that there are at least $k' - k + 2$ eventual leaders (w.r.t. *isLeader*$'$ outputs) in P_0. If not, then the set of eventual leaders in P_0 can be completely covered by some subset S_j in the round-robin sequence. By our construction above, for every S_j's turn, there is one process in $P_0 \setminus S_j$ that sets its *isLeader*$'$ to *True*. Since there are an infinite number of turns for S_j, we conclude that there must be at least one eventual leader in $P_0 \setminus S_j$. Hence, we cannot have only at most $k' - k + 1$ eventual leaders in P_0.

However, when we count the total number of eventual leaders (w.r.t. *isLeader*$'$ outputs) in run R, we have at least $k' + 1$ eventual leaders, since each p_i with $1 \leq i \leq k - 1$ is an eventual leader and P_0 has at least $k' - k + 2$ eventual leaders. This violates the property of $\Omega''_{k'}$ (in particular properties $(\Omega''1)$ and $(\Omega''5)$). □

Lemma 2 *In model $\mathcal{A}$, for any $k \geq 2$, any $k' \geq k$, and any $n \leq 2k' - k + 1$, Π^S_k can be transformed into $\Omega''_{k'}$.*

Proof. Let (*isLeader*, *lbound*, *Quorum*, *cid*) be the output of a failure detector $\mathcal{D}$ in Π^S_k. We describe a transformation algorithm T that produce the output (*isLeader*$'$, *lbound*$'$) satisfying the constraints of $\Omega''_{k'}$. In T, we set *lbound*$'$ $= k'$ for all processes, which satisfies properties $(\Omega''1)$ and $(\Omega''2)$ of $\Omega''_{k'}$ (see page 3 for property definition). For the *isLeader*$'$ output of $\mathcal{D}'$ on a process p, T sets it to *False* if the *lbound* output on p is 0, otherwise it sets it as the following.

Intuitively, we try to use a transformation algorithm similar to the one in Theorem 2 to construct $\Omega''_{\lceil |P_i|/2 \rceil}$ in every leaf component P_i in the partition tree Γ. But the problem is the processes do not know which processes are in the same leaf component in advance. So T needs to estimate the membership of each component first. To do so, each process p maintains a set P_p representing p's view of processes in its partitioned component. Initially, $P_p = \{p\}$. Process p periodically broadcasts its *Quorum*, P_p and *cid* to all other processes in the system. Suppose q receives a message (p.*Quorum*, P_p, *cid*) from process p. If p.*cid* $\not\equiv$ q.*cid*, the message is discarded since p and q are in different components. Otherwise, q updates P_q as $P_q = P_q \cup p$.*Quorum* $\cup P_p$. When the *cid* output of Π^S_k on p changes, the component containing p splits. In this case, we reset the content of P_p to be $\{p\}$ to re-estimate the component membership. Suppose P_i is the leaf component containing p, i.e. p.*cid* $\equiv P_i$ eventually. After p.*cid* stabilized, p only adds processes into P_p and never adds processes in other components, P_p can only grow but never exceed P_i. Hence, P_p will stabilize eventually. Also, because p broadcasts its P_p to everyone else in P_i, eventually all correct processes in P_i will have the same view as P_p. Therefore, all correct processes in the same partitioned component will have the same view on their component. Such a view must contain all correct processes, because they send messages to each other repeated. The view can contain some (possibly not all) faulty processes in the component as well.

Now we choose a total order "$\leq$" among the subsets of P such that $|A| \leq |B|$ implies $A \leq B$, and then use *Quorum* in the output of $\mathcal{D}$ to calculate the *Leaders* set. Since P_p changes over time, we cannot set the initial value for *Leaders* as in Theorem 2. So we introduce a new variable *minQ*, which records the smallest *Quorum* (regarding the total order "$\leq$") p has ever seen during the period its *cid* stay unchanged.

Initially, $p.\mathit{minQ} = H(p, 0).\mathit{Quorum}$. Process p periodically checks its *Quorum* and *cid* output. If *cid* changes or *Quorum* $\leq$ *minQ*, p sets its *minQ* to the *Quorum* output. Process p also periodically broadcasts its *minQ* (instead of *Leaders*) and *cid* value to all other processes. When receiving a message m, process p discards the messages from the process in other partitioned components by checking whether $m.\mathit{cid} \equiv p.\mathit{cid}$. If $m.\mathit{cid} \equiv p.\mathit{cid}$, p compares $m.\mathit{minQ}$ with $p.\mathit{minQ}$. If $m.\mathit{minQ} \leq p.\mathit{minQ}$, p sets $p.\mathit{minQ}$ to $m.\mathit{minQ}$. Let $B_p = \max\{A \subseteq P_p \mid |A| \leq \lceil |P_p|/2 \rceil\}$, where P_p is p's current view of its partitioned component. And we always set $p.\mathit{Leaders} = \min(p.\mathit{minQ}, B_p)$, i.e. $p.\mathit{Leaders}$ is the alias of $\min(p.\mathit{minQ}, B_p)$. Then, $p.\mathit{isLeader}'$ is set to *True* if and only if $p \in p.\mathit{Leaders}$.

Now, we prove the correctness of T.

First, we prove that in leaf component P_i, there are at most $\lceil |P_i|/2 \rceil$ eventual leaders (the leaders' *isLeader$'$* remains *True* eventually). For all process $p \in P_i$, P_p eventually stablizes and converges to the same value. Let this final view be Q_i. Hence, B_p also stabilizes and converge to the same value, say B_i. On the other hand, the broadcast and calculation of *minQ* ensures that $p.\mathit{minQ}$ also stablizeds and converges to the same value, say minQ_i. Therefore, $p.\mathit{Leaders}$ stablizes and converge to a value, say $\mathit{Leaders}_i$. Because we reset the value of P_p and *minQ* every time *cid* changes, and also all the messages from other leaf components are filtered out, we have $Q_i \subseteqq P_i$ and $\mathit{minQ} \subseteqq P_i$. Since $\mathit{Leaders}_i \leq B_i$, it must be true that $|\mathit{Leaders}_i| \leq |B_i| = \lceil |Q_i|/2 \rceil \leq \lceil |P_i|/2 \rceil$. So at most $\lceil |P_i|/2 \rceil$ processes in P_i set their *isLeader$'$* to *True*. Also, $(\Omega''3)$ of $\Omega''_{k'}$ (see page 3 for property definition) holds due to the stability of set *Leaders*.

Then, we prove that at least one correct process will be one of the eventual leaders (w.r.t. *isLeader$'$* outputs). Note that the above argument cannot guarantee this property, because the *Quorum* output of $\mathcal{D}$ in leaf components does not have the liveness property of Σ, making the set *Leaders* composed of purely faulty processes. But Π_k^S ensures the existance of a live leaf component P_j, in which the *Quorum* output satisfies all of the specification of Σ localized in P_j. If $\mathit{Leaders}_j = \mathit{minQ}_j$, a correct process is in $\mathit{Leaders}_j$ because minQ_j intersects with the final *Quorum* output containing only correct processes and hence there is at least one correct process in minQ_j. If, on the other hand, $\mathit{Leaders}_j \neq \mathit{minQ}_j$, we have $\mathit{Leaders}_j = B_j$. Because Q_j contains all the correct process, we know $\mathit{Quorum} \subseteqq Q_j$ eventually. Since $B_j < \mathit{minQ}_j < \mathit{Quorum}$ and $|B_j| = \lceil |Q_i|/2 \rceil$, $|\mathit{Quorum}| \geq \lceil |Q_i|/2 \rceil$. So $B_j \cap \mathit{Quorum} \neq \emptyset$ for all *Quorum* output containing correct processes only. As a result, there is still a correct process in $\mathit{Leaders}_j$. So, $(\Omega''4)$ of $\Omega''_{k'}$ (see page 3 for property definition) holds.

Finally, we prove the number of eventual leaders elected by our algorithm is not larger than k', that is, $(\Omega''5)$ of $\Omega''_{k'}$ holds. Suppose the leaf components of failure detector $\mathcal{D}$ are $P_1, \cdots, P_l$ while $m_i = |P_i| (\sum_i m_i = n)$. We can omit the components in which *lbound* is always 0, because there are no leaders elected in such components. So we can suppose $l \leq k$. Then the number of eventual leaders $\leq \lceil m_1/2 \rceil + \cdots + \lceil m_l/2 \rceil \leq \frac{m_1+1}{2} + \cdots + \frac{m_l+1}{2} = \frac{n+l}{2} \leq \frac{n+k}{2}$. So, if $2k' \geq n + k$, we are done. If $2k' = n + k - 1$, n and k must have different parity. So, either $l < k$, or there exists at least one even m_i. Then the number of leaders $\le \lceil m_1/2 \rceil + \cdots + \lceil m_l/2 \rceil \le \frac{n+k-1}{2} = k'$. So, for $2k' \ge n+k-1$, we have $(\Omega''5)$ of $\Omega''_{k'}$ holds. □

Theorem 5 *Π_2^S cannot be transformed into Π_k in model $\mathcal{A}$.*

Proof. Suppose, for a contradiction, that there exists a transformation algorithm T from Π_2^S to Π_k. Let $(\mathit{isLeader}, \mathit{lbound}, \mathit{Quorum}, \mathit{cid})$ be the output of a failure detector $\mathcal{D}$ in Π_2^S, and $(\mathit{isLeader}', \mathit{lbound}', \mathit{Quorum}')$ be the output of a failure detector $\mathcal{D}'$ in Π_k. For the *cid* output, we choose to use integer values, and $\equiv$ is defined to be normal $=$ relation among integers.

We construct a run R of T with failure detector $\mathcal{D}$, in which all processes are correct, and all *lbound* outputs are always 1. Initially, we set *Quorum* of all processes to the full set P, and set *cid* to 0. We choose one process p_1 and set its *isLeader* to *True*, and set *isLeader* outputs of all other processes to *False*. We run algorithm T with the above failure detector outputs from $\mathcal{D}$. Since the above outputs of $\mathcal{D}$ satisfy the properties of Π_k^S when the partition tree has only one root node (i.e., no partition at all), the run of T should produce legitimate outputs of $\mathcal{D}'$ satisfying the properties of Π_k.

Let $Q_1, \ldots, Q_n$ be the outputs of *Quorum*$'$ on processes $p_1, \ldots, p_n$ respectively, generated by T at some arbitrary time t_0 during the run. There are two possible cases. In the first case, there exists Q_j and p_i such that $p_i \ne p_j$ and $p_i \in Q_j$. In this case, we split P into two sets P_1 and P_2 such that $p_i \in P_1$ and $p_j \in P_2$. This corresponds to a partition tree Γ with P as the root and P_1, P_2 as the children of P, and the split history S such that $S(p,t) = P$ for all p and all $t \le t_0$ and $S(p,t) = P_i$ for all $p \in P_i$ and all $t > t_0$. After time t_0, we adjust the output of $\mathcal{D}$ as follows. For process p_i, we set its *isLeader* to *True*, its *Quorum* to $\{p_i\}$, and its *cid* to 1. For all other processes in P_1, we set their *isLeader* to *False*, their *Quorum* to P_1, and their *cid* to 1. For process p_j, we set its *isLeader* to *True*, its *Quorum* to $\{p_j\}$, and its *cid* to 2. For all other processes in P_2, we set their *isLeader* to *False*, their *Quorum* to P_2, and their *cid* to 2.

Note that we only split once with two components after the split. Each component has one eventual leader, which is within its eventual *lbound* value 1, and the sum of the maximum *lbound* values of two components is 2. Other properties of Π_2^S are also easy to verify, so we know that the outputs of $\mathcal{D}$ satisfy all properties of Π_2^S. We delay all messages sent to p_i and all messages sent to p_j and run algorithm T. Process p_i cannot distinguish this run with another run in which all other processes crash at time $t_0 + 1$. In that run $\{p_i\}$ is the only live component, so eventually the *Quorum*$'$ output of p_i must be $\{p_i\}$. Similarly, we can argue that eventually the *Quorum*$'$ output of p_j must be $\{p_j\}$.Once we obtain these *Quorum*$'$ outputs, we can deliver all messages that have been delayed.

In run R, since the outputs of $\mathcal{D}$ satisfy all properties of Π_k^S, the outputs of $\mathcal{D}'$ generated by T should satisfy all properties of Π_k. Since in R we obtain two *Quorum*$'$ outputs $\{p_i\}$ and $\{p_j\}$, it implies that p_i and p_j cannot be in the same statically partitioned component. However, in run R we also have a *Quorum*$'$ output Q_j on process p_j that contains p_i, which means p_i and p_j must be in the same statically partitioned component. Therefore, we have reached a contradiction in this case.

We now consider the second case, in which $Q_j = \{p_j\}$ for all $j = 1, \ldots, n$. This implies that the static partition for failure detector $\mathcal{D}'$ must be a partition with all singleton components: $\{\{p_1\}, \ldots, \{p_n\}\}$. In this case, we construct run R such that we see at least $k+1$ processes having their *lbound*$'$ set to at least 1 at some point in time. Since the partition is singleton, this means that the sum of maximum *lbound*$'$ outputs in all partitioned components is at least $k+1$, violating property $(\Pi\Omega 1)$ of Π_k.

We now construct run R after time t_0. We let the algorithm T run until we see some process q_1 with *lbound*$'$ output being at least 1 at some time $t_1 > t_0$. This must happen because as we described earlier the outputs of $\mathcal{D}$ is legitimate with a partition tree that has only one root node, and thus there must be a live component (w.r.t. $\mathcal{D}'$) in which the *lbound*$'$ outputs eventually stabilize to a value of at least 1.

Let $S_1 = P \setminus \{q_1\}$. After time t_1, we change the outputs of failure detector $\mathcal{D}$ as follows. The *Quorum* output of all processes are changed to S_1, and we choose one process in S_1 to set its *isLeader* to *True*, and we set the *isLeader* of all other processes in P to *False*. The outputs of *lbound* remain to be 1 and the outputs of *cid* remain to be 0. We delay all messages sent to processes in S_1 from q_1 and let T run. Processes in S_1 cannot distinguish this run with another run in which q_1 crashes at time $t_1 + 1$. In the latter run, the single component P is still a live component (w.r.t. $\mathcal{D}$) so eventually algorithm T should set *lbound'* of some process $q_2 \in S_1$ to at least 1 at time t_2.

We then let $S_2 = S_1 \setminus \{q_2\}$ and repeat the above procedure on S_2. We can repeat the procedure until we find sets $S_1, \ldots, S_k$, processes $q_1, \ldots, q_k, q_{k+1}$, and time points $t_1, \ldots, t_k, t_{k+1}$, such that $q_{j+1} \in S_j$, $S_{j+1} = S_j \setminus \{q_{j+1}\}$ and the *lbound'* output of q_j at time t_j is set to at least 1. After time t_{k+1}, we have no more change to the outputs of $\mathcal{D}$.

It is easy to verify that the outputs of $\mathcal{D}$ in run R satisfy all properties of Π_2^S with a single component P (as an example, all *Quorum* outputs intersect on processes in S_k). Therefore, algorithm T should generate outputs that satisfy Π_k. However, by the above construction, we find at least $k + 1$ processes with *lbound'* values of at least 1 at some point in time, and since we already know that the partitioned components for $\mathcal{D}'$ are all singleton, we know that the sum of the maximum *lbound'* values of all partitioned components are at least $k + 1$. This violates property $(\Pi\Omega 1)$ of Π_k.

Therefore, in both cases we reach contradictions. The theorem thus holds. $\square$

Theorem 6 *For any $k \geq 2$ and any $n \geq 2k - 1$, failure detector classes $\Omega_k'' \times \Sigma$, Π_k, and Π_k^S cannot be used to solve $(k-1)$-set agreement in model $\mathcal{A}$. Together with Theorem 11, this implies that classes $\Omega_k'' \times \Sigma$, Π_k, and Π_k^S cannot be transformed into classes $\Omega_{k-1}'' \times \Sigma$, Π_{k-1}, and Π_{k-1}^S.*

Proof. Theorem 1 of [14] shows that in model $\mathcal{A}[Maj]$ with $n > 2k$, if class Ω_z can be used to solve k-set agreement, then $z \leq k$. In our case, if $\Omega_z'' \times \Sigma$ can be used to solve $(k-1)$-set agreement, we know that Ω_z'' can be used to solve $(k-1)$-set agreement in $\mathcal{A}[Maj]$ (because $\mathcal{A}[Maj]$ is stronger than $\mathcal{A}[\Sigma]$). By our result in [5], Ω_z'' is equivalent to Ω_z. Therefore, we know that $z \leq k - 1$. This means $\Omega_k'' \times \Sigma$ cannot solve $(k-1)$-set agreement in $\mathcal{A}$ with $n \geq 2k - 1$. Since $\Omega_k'' \times \Sigma$ is stronger than Π_k and Π_k^S, we know that Π_k and Π_k^S cannot solve $(k-1)$-set agreement either. $\square$

Theorem 7 *In model $\mathcal{A}$, when $k \geq 2$, we have*
(1) for any $n \geq 2k - 1$, $\Omega_k'' \times \Sigma$ cannot be transformed into Π_{k-1}^S; and
(2) for any $n > k$, Π_k cannot be transformed into Π_{k-1}^S.

We prove this theorem in the following two lemmas.

Lemma 3 *In model $\mathcal{A}$, for any $k \geq 2$ and any $n \geq 2k - 1$, $\Omega_k'' \times \Sigma$ cannot be transformed into Π_{k-1}^S.*

Proof. Suppose, for a contradiction, that there exists an algorithm T that transforms any failure detector $\mathcal{D}$ in $\Omega_k'' \times \Sigma$ to a failure detector $\mathcal{D}'$ in Π_{k-1}^S. Let $(\mathit{isLeader}, \mathit{lbound}, \mathit{Quorum})$ be the output of a failure detector $\mathcal{D}$ in $\Omega_k'' \times \Sigma$ and $(\mathit{isLeader}', \mathit{lbound}', \mathit{Quorum}', \mathit{cid}')$ be the output of a failure detector $\mathcal{D}'$ in Π_{k-1}^S generated by algorithm T.

Let F be a failure pattern, H' be the failure detector history of $\mathcal{D}'$ generated by T under failure pattern F, Γ be the partition tree for H', and S be the partition split history of Γ for H'. Let $\{P_1, \cdots, P_m\}$ be a partition of P. We define $P_i.\mathit{lbound}(t) = \max\{H'(p, t').\mathit{lbound} \mid t' \leq t, p \in P_i \setminus F(t')\}$. We define $A(P_i, t) = \{p \in P_i \setminus F(t) \mid H'(p, t).\mathit{isLeader}' = \mathit{True}\}$. 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.

If the partition $\{P_1, \cdots, P_m\}$ is a partition of P that occurs in H' for $\mathcal{D}'$, i.e., it is $\{S(p, t) \mid p \in P\}$ for some t, it is straightforward to see that $P_i.\mathit{lbound}(t) \leq \sum_{Q_i \subseteq P_i, Q_i \in L(\Gamma)} \mathit{lbound}(Q_i)$ for any time t, where $L(\Gamma)$ is the set of leaf nodes in Γ. Then by property $(\Pi^S\Omega 1)$ of Π_{k-1}^S, we know that for any time t, $\sum_i P_i.\mathit{lbound}(t) \leq k-1$. Thus we have $|A(t)| \leq k-1$. So $|P \setminus A(t)| \geq n-k+1 \geq (n+1)/2$. That is, $P \setminus A(t)$ contains a majority of processes. Therefore, if we always set *Quorum* outputs to such sets, it is guaranteed that all *Quorum* outputs intersect.

We now construct an infinite sequence of runs $R_0, R_1, \ldots$ by the following inductive process. In all these runs, we always set *lbound* $= k$ on all processes, and set a fixed k processes $p_1, \ldots, p_k$ with *isLeader* $=$ *True*. All other processes have *isLeader* set to *False*.

Run R_0: In this run, no processes are crashed. So we set *Quorum* $= P$. We run the algorithm T with the above outputs of $\mathcal{D}$. Then, find a time t_0 such that for each pair of processes p and q, $H'(p,t_0).\mathit{Quorum} \cap H'(q,t_0).\mathit{Quorum} \neq \emptyset$ implies $H'(p,t_0).\mathit{cid}' \equiv H'(q,t_0).\mathit{cid}'$. In other words, all processes have a consistent view of system partition.

Run R_1: R_1 runs exactly the same as in R_0 until time t_0. At time t_0, we can define a partition scheme by the *cid* output on all processes. 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. At time t_0+1, we crash all processes in $A(t_0)$ and then set *Quorum* $= P \setminus A(t_0)$ for all the other processes. 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, all messages sent by time t_0 have been delivered in R_1, and the system partition views on all correct processes are consistent. If there are some undelivered messages for processes in $A(t_0)$, they are not dropped because we may need to deliver them in the subsequent run R_2, which is built based on 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}, all messages sent before t_{i-2} are are delivered by t_{i-1}, and all processes not in $A(t_{i-2})$ have consistent partition views at time 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 channel states are exactly the same as in run R_{i-1}.

During the execution between $t_{i-2}+1$ and t_{i-1}, it is possible for the partitioned components to split further. If there are component splits, i.e. $\exists p \in P \setminus A(t_{i-2}), H'(p,t_{i-2}).\mathit{cid}' \not\equiv H'(p,t_{i-1}).\mathit{cid}'$, we let all the processes in P execute after t_{i-1}. If there are no splits, i.e. $\forall p \in P \setminus A(t_{i-2}), H'(p,t_{i-2}).\mathit{cid}' \equiv H'(p,t_{i-1}).\mathit{cid}'$, we calculate $A(t_{i-1})$ in a similar manner as described in R_1. Then we crash the processes in $A(t_{i-1})$ at $t_{i-1}+1$, set *Quorum* outputs of the rest of the processes to $P \setminus A(t_{i-1})$, 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 $t_{i-1}+1$, all messages sent by time t_{i-1} have been delivered, all correct processes have consistent view on system partition.

Since there are a finite number of processes in P, $\mathcal{D}'$ is allowed to split partitioned components for a finite number of times. So there exists u, in all runs R_l for $l \geq u$, no component splits happen after t_{u-2}. Therefore, all the processes in P fall into the leaf nodes of the partition tree of $\mathcal{D}'$ in all of the runs R_l after t_{u-2} for $l \geq u$.

So far we have constructed a 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 messages, the failure detector history, and the sequence of steps of run R_∞ be identical to the run R_i until time t_{i-1}. And the partition tree Γ and partition split history S of run R_∞ is the same as run R_l for $l \geq u$. We need to show that R_∞ is still a legitimate run of algorithm T with some failure detector in $\Omega''_k \times \Sigma$.

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 R. First, we need to show that the the failure pattern F derived above does not make the output of $\mathcal{D}$ violate any property of $\Omega''_k \times \Sigma$. For all faulty processes in R_∞, by their definition above, we know that there must exists some $A(t_j)$ that contain all these faulty processes. We already know that $|A(t_j)| \leq k - 1$, so at least one process in $p_1, \ldots, p_k$ is correct in R_∞. This process always has its *isLeader* set to *True*, so it is an eventual leader, i.e., $(\Omega''4)$ holds. All other properties of Ω''_k holds trivially. For the *Quorum* outputs, they are always in the form of $P \setminus A(t_j)$, so by our earlier argument, they are majority quorums and thus must intersect with one another. Moreover, if p is not a correct process, then p crashes at some time $t_{j_p-1} + 1$, and by our construction of runs p will not be in the *Quorum* outputs of any processes after $t_{j_p-1} + 1$. So the *Quorum* outputs satisfy the properties of Σ.

Second, we need to verify that in run R_∞, all correct processes take an infinite number of steps and all messages sent to the correct processes are eventually delivered. 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_∞. Similarly, suppose m is a message sent to a correct process p in R_∞ at time t. Then 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, all messages sent to correct processes by time t_{j-1} are delivered by time t_j. Thus we know that in R_∞ message m is delivered to p by time t_j.

Therefore, by the above arguments, we know that R_∞ is a legitimate run of algorithm T with a failure detector $\mathcal{D}$ in $\Omega''_k \times \Sigma$. Then we know that T should generate correct outputs of $\mathcal{D}'$ in Π^S_{k-1}. This means that eventually there is a leaf 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. □

Lemma 4 *In model $\mathcal{A}$, for any $k \geq 2$ and any $n > k$, Π_k cannot be transformed into Π^S_{k-1}.*

Proof. Suppose, for a contradiction, that there exists an algorithm T that transforms any failure detector $\mathcal{D}$ in Π_k into a failure detector $\mathcal{D}'$ in Π^S_{k-1}. Let $(\mathit{isLeader}, \mathit{lbound}, \mathit{Quorum})$ be the output of $\mathcal{D}$ and $(\mathit{isLeader}', \mathit{lbound}', \mathit{Quorum}', \mathit{cid}')$ be the output of $\mathcal{D}'$.

We construct a run R of T with failure detector $\mathcal{D}$ as follows. Let $\{P_1, \ldots, P_k\}$ be a static partition of P, and let $p_i \in P_i$ be some process in component P_i for every i. In run R all processes are correct. We set *lbound* always to 1 on all processes. For each process in P_i, its *Quorum* output is always P_i. For process $p_i \in P_i$, its *isLeader* is always *True*, while for all other processes in P_i, their *isLeader* values are always *False*. It is easy to check that with these output values, $\mathcal{D}$ is a valid failure detector in Π_k, with the static partition $\{P_1, \ldots, P_k\}$.

In run R, we delay all messages sent across different partitions first. For processes in component P_i, they cannot distinguish this run with another run R_i in which only processes in P_i are correct and all other processes crash at the beginning of the run. In run R_i, the outputs of $\mathcal{D}$ for all processes in R_i are the same as in run R. It is easy to check that the output of $\mathcal{D}$ in this run still satisfy the properties of Π_k, with P_i as the only live component. Therefore, in run R_i, the output of $\mathcal{D}'$ generated by algorithm T should satisfy the properties of Π^S_{k-1}. Then there exists a leaf node partition of P, such that at least one leaf component P'_i in the partition is a live component (w.r.t. $\mathcal{D}'$). This implies that there exists a correct process $p'_i \in P'_i$ such that eventually (a) the *lbound*$'$ output on p'_i is the same as other correct processes; (b) the *lbound*$'$ output is at least 1 since there is at least one eventual leader in the live component P'_i; (c) the *Quorum*$'$ output on p'_i contains only correct processes. Since only processes in P_i are correct in run R_i, we know that $p'_i \in P_i$, and we find a time t_i at which the *lbound*$'$ output of p'_i is at least 1 and the *Quorum*$'$ output of p'_i is contained in P_i.

The above argument holds for all $i = 1, \ldots, k$. Then we complete run R by selecting a time $t = \max(t_1, \ldots, t_k)$ such that after time t, all delayed messages across different components are delivered, and we do not further control the run.

Therefore, we have a legitimate run R of algorithm T with failure detector $\mathcal{D}$. In run R, we find k processes $p'_1, \ldots, p'_k$ such that their *Quorum*$'$ outputs at some time do not intersect with one another (since at a time *Quorum*$'$ output on p'_i is contained in P_i, and $\{P_1, \ldots, P_k\}$ is a partition). By property $(\Pi\Sigma2)$ of Π_{k-1}, processes $p'_1, \ldots, p'_k$ must be in different leaf node components in R.Thus, we have at least k leaf node components in R w.r.t $\mathcal{D}'$.

The maximum *lbound*$'$ value in each of these k components must be at least 1, since the *lbound*$'$ output of p'_i at time t_i is at least 1. Then we know that the sum of maximum *lbound*$'$ outputs among all leaf node components is at least k. This result contradicts to property $(\Pi\Omega1)$ of Π^S_{k-1}, which requires that the sum of maximum *lbound*$'$ outputs among all leaf node components be at most $k-1$. □

Theorem 9 *When $n \geq 4k+2$, Π_2 cannot be transformed into Ω''_k in model $\mathcal{A}[\Sigma]$ or $\mathcal{A}[Maj]$.*

Proof. We only need to consider model $\mathcal{A}[Maj]$ since it is stronger than $\mathcal{A}[\Sigma]$. Suppose, for a contradiction, that there exists an algorithm T that transforms any failure detector $\mathcal{D}$ in Π_2 to a failure detector $\mathcal{D}'$ in Ω''_k in model $\mathcal{A}[Maj]$. Let (*isLeader*, *lbound*, *Quorum*) be the output of $\mathcal{D} \in \Pi_2$, and let (*isLeader*$'$, *lbound*$'$) be the output of $\mathcal{D}' \in \Omega''_k$.

We partition the set of processes P into two components P_1 and P_2, with n_1 and n_2 processes respectively, such that $n = n_1 + n_2$, $n_1 \geq 2k+1$ and $n_2 \geq 2k+1$. We will construct a run R of algorithm T with failure detector $\mathcal{D}$, in which all processes are correct, and at any time, $|Quorum| = n_1 - k$ on all processes in P_1, and $|Quorum| = n_2 - k$ on all processes in P_2.

It is easy to verify that any two *Quorum* outputs in P_1 intersect, and any two *Quorum* outputs in P_2 intersect. Thus the outputs of $\mathcal{D}$ satisfy the quorum related properties of Π_2, namely $(\Pi\Sigma1)$, $(\Pi\Sigma2)$, and $(\Pi\Sigma3)$. We choose an arbitrary process $p_1 \in P_1$, set *isLeader* always to *True* on p_1 and always to *False* on all processes in $P_1 \setminus \{p_1\}$. We set *lbound* always to 1 for all processes in the two components.

With the above constraints, it is easy to verify that the outputs of $\mathcal{D}$ in both components satisfy all the safety properties of Π_2, while the outputs of $\mathcal{D}$ in component P_1 satisfy all the liveness properties of Π_2, i.e., P_1 is the live component (w.r.t. $\mathcal{D}$). Therefore, the run of algorithm T with the above outputs of $\mathcal{D}$ should produce outputs of $\mathcal{D}'$ that satisfy the specification of Ω''_k. This is true no matter how other values, in particular the actual values of *Quorum* outputs, and the *isLeader* values for processes in component P_2, are set. We manipulate these values to construct run R, as described below.

For any time t and any time $t' > t$, let $L(t, t') = \{p \in P \mid$ *isLeader*$'$ on p be *True* at some time t'' with $t < t'' \leq t'\}$. Initially we set the outputs of $\mathcal{D}$ to any values as long as they satisfy the above constraints. By the specification of Ω''_k, we know that there exists a time t such that for all $t' > t$ $|L(t, t')| \leq k$. Let the algorithm T run passing time t and to some time $t' > t$. Choose a set $B_1 \subseteq P_1$ and a set $B_2 \subseteq P_2$ such that $B_1 \cap L(t, t') = \emptyset$, $B_2 \cap L(t, t') = \emptyset$, $|B_1| = n_1 - k$, and $|B_2| = n_2 - k$. Since $|P_1| = n_1$ and $|P_2| = n_2$ and $|L(t, t')| \leq k$, it is possible to choose such B_1 and B_2.

Set *Quorum* to B_1 for all processes in component P_1 and *Quorum* to B_2 for all processes in component P_2 after time t', so *Quorum* outputs satisfy the contraints described earlier. Choose an arbitrary process p_2 in B_2 and set *isLeader* $=$ *True* on p_2 and *False* on all other processes in component P_2. Then we delay the messages sent to processes in $B_1 \cup B_2$ from processes not in $B_1 \cup B_2$. For any process in $B_1 \cup B_2$, it cannot distinguish run R from another run R' in which all processes in $P \setminus (B_1 \cup B_2)$ crash at time t'.

Note that $B_1 \cup B_2$ contains a majority of processes (because $2|B_1 \cup B_2| = 2(n_1 - k + n_2 - k) = 2n - 4k > n$), so run R' is possible in model $\mathcal{A}[\mathit{Maj}]$. In run R', B_2 is a live component (w.r.t. $\mathcal{D}$). So, there exists time $t'' > t'$ at which at least one process $p \in B_1 \cup B_2$ sets its *isLeader*$'$ to *True*. Therefore, in run R we can find a time $t'' > t'$ such that $p \in L(t, t'') \setminus L(t, t')$, which means $|L(t, t'')| > |L(t, t')|$ (note that run R does not crash any processes). If $|L(t, t'')| \leq k$, we can repeat the above procedure to set some processes not in $L(t, t'')$ as a leader. Therefore, eventually, we will find a time $t'' > t$ such that $|L(t, t'')| > k$.

Hence, in run R whenever there are times t and $t' > t$ such that $|L(t, t')| \leq k$, we can repeat the above procedure so that there is a time $t'' > t'$ and $|L(t, t'')| > k$. This implies that the number of eventual leaders (w.r.t. *isLeader*$'$) in run R cannot be at most k, contradicting the specification of Ω''_k. □

Theorem 10 *When* $n \geq 4k + 2$, Π^S_2 *cannot be transformed into* Π_k *in model* $\mathcal{A}[\Sigma]$ *or* $\mathcal{A}[\mathit{Maj}]$.

The basic idea of the proof is that, suppose there is such a transformation T, then let T run to see how the outputs of Π_k partition P. There will be two possible partition cases:

1. There is at most one component with more than one process in it.
2. There are multiple components containing more than one process.

In case 1, we split P into two components with almost equal size regarding to the constraints of Π^S_2. Then we crash some processes and force the algorithm T to actually implement Ω''_{k-m} on the rest of correct processes. At last we stated that the number of correct processes exceeds the bound mentioned in Theorem 9, and the contradiction is reached.

In case 2, we choose the smallest component with more than one process, and divide it into two sets A and B. We first select a process in A as an eventual leader w.r.t. Π^S_2 and crash all processes in B, and use the proof technique in Theorem 7 (1) to try to construct a run of T that does not generate any live component w.r.t. Π_k. The only case the proof cannot continue as in the proof of Theorem 7 (1) is when p is always selected as an eventual leader in a live component w.r.t. Π_k. In this case, p's *Quorum* output of Π_k must be contained in set A. Then we select a process q in set B, crash all processes in set A, and repeat the argument as in the proof of Theorem 7 (1). Again, the only case where the proof cannot continue is when q is always selected as an eventual leader in a live component w.r.t. Π_k. However, when this happens, q's *Quorum* output of Π_k must be contained in set B, but B does not intersect with A. So we find two processes p and q in the same static component of Π_k but their *Quorum* outputs do not always intersect. Therefore, we reach a contradiction, and this final contradiction resembles the proof technique used in Theorem 5.

Proof. We only need to consider model $\mathcal{A}[\mathit{Maj}]$. Suppose, for a contradiction, that there exists a transformation algorithm T from Π^S_2 to Π_k. Let $(\mathit{isLeader}, \mathit{lbound}, \mathit{Quorum}, \mathit{cid})$ be the output of a failure detector $\mathcal{D}$ in Π^S_2, and $(\mathit{isLeader}', \mathit{lbound}', \mathit{Quorum}')$ be the output of a failure detector $\mathcal{D}'$ in Π_k. For the *cid* output, we choose to use integer values, and $\equiv$ is defined to be normal $=$ relation among integers.

Initially(at time 0), we set *Quorum* of all processes to the full set P, set *cid* to 0, and set *lbound* to 1. We choose one process p_1 and set its *isLeader* to *True*, and set *isLeader* outputs of all other processes to *False*. We run algorithm T with the above failure detector outputs from $\mathcal{D}$. Since the above outputs of $\mathcal{D}$ satisfy the properties of Π^S_2 when the partition tree has only one root node (i.e., no partition at all), the run of T should produce legitimate outputs of $\mathcal{D}'$ satisfying the properties of Π_k.

By reading the outputs of *Quorum*$'$ on processes $p_1, \ldots, p_n$ respectively, we can know the partition of $\mathcal{D}'$: $P_1, \cdots, P_l$. There are two possible cases.

Case 1 In this case, there exists at most one partition component which contains at least two processes. Suppose the partition is $\{P_1, \cdots, P_l, Q\}$ where $P_i = \{p_i\}$ and $|Q| \geq 1$. At time $t_1 = 1$, we split P into two parts A and B with $|A| \geq 2k+1$ and $|B| \geq 2k+1$. For every process in A, we set $\mathcal{D}$'s output (*Quorum*, *cid*, *lbound*) to be $(A, 1, 1)$. For the processes in B, we set the output to be $(B, 2, 1)$. It is obvious that either $Q \cap A$ or $Q \cap B$ is not empty. Without loss of generality, suppose $Q \cap A \neq \emptyset$. We pick an arbitrary process $q \in Q \cap A$ and set its *isLeader* to *True*. The *isLeader* of processes in $P \setminus \{q\}$ is set to *False*. Then we let T to execute.

During the execution, we check the *lbound'* output of $\mathcal{D}'$ on the set of processes $\{p_i\}(1 \leq i \leq l)$. If at a time $t > 1$, we find $H'(p_i, t)$.*lbound'* ≥ 1 we crash process p_i, and modify the *Quorum* output of $\mathcal{D}$ to exclude p_i. Suppose we crash m processes eventually, we will have $m < k$. If $m \geq k$, since T is correct, one of the correct process, say p' will set its *lbound'* ≥ 1 at time t'. Now we can construct another one R', of which the only difference to R is that the processes are not crashed, but kept from execution. Since the algorithm is determinstic, and the process and channel state are same in R' and R, at time t' there will be at least $k+1$ processes each in a different component set their *lbound* to be at least 1. This violates the property $(\Pi\Omega 1)$ of Π_k. So we have $m \leq k$.

Now we turn back to R. For each correct process p', if $p' \in P_i$ for some $i, 1 \leq i \leq l$, we know p'.*lbound* is set to 0. Therefore, all the P_is not crashed do not have liveness. So Q must be a live component. With an argument similar to the one for $m < k$, we know the *lbound* output of processes in component Q is at most $k - m$. If T can still produce correct output for $\mathcal{D}'$, the output can be easily transformed to a failure detector in Ω''_{k-m}. This is because we can ignore the output of p_i and broadcast the *lbound'* of processes in Q. Let C be the set of crashed process, $A' = A \setminus C$, and $B' = B \setminus C$. We have $|A'| \geq 2k + 1 - m \geq 2(k - m) + 1$ and $|B'| \geq 2k+1-m \geq 2(k-m)+1$. Using a similar run construction shown in the proof of Theorem 9, we know that failure detectors in Ω''_{k-m} cannot be implemented. This is contradictory to the fact that $\mathcal{D}'$ can be transformed to a failure detector in Ω''_{k-m}.

Case 2 In this case, there exists at least two partition components which contains at least two processes. Suppose P_1 and P_2 are two components with at least two processes and $|P_1| \leq |P_2|$. We divide P_1 into two equal parts P_{11} and P_{12} with $||P_{11}| - |P12|| \leq 1$. Note $|P_{11}| \geq 1$ and $|P_{12}| \geq 1$. We define $A(t) = \{p |\ p \in$ a quasi-live component at time t and $H'(p, t)$.*isLeader'* $=$ *True*$\}$. The definition is the same as that of $A(t)$ in the proof of Lemma 3.

We then construct two classes of infinitely runs to reach contradiction. Step 1, at time $t_1 = 1$, we crash all processes in P_{11}. We set *Quorum* $= P/P_{11}$, set *cid* to 0, and set *lbound* to 2 for all other processes. We choose one process p_2 in P_{12} and set its *isLeader* to *True*, and set *isLeader* outputs of all other processes to *False*. This is run R_1^1. To construct run R_i^1, we always crash P_{11} at time 1 and treat other processes using the same method in the proof of Lemma 3. If, at time t_i, we find $p_2 \in A(t_i)$, we stop constructing R_i^1 and go to step 2. If we never go to step 2, we will have an infinite number of runs $\{R_i^1\}$. Note that $p_2 \notin A(t_i)$ for all i, so in each run R_i^1, p_2 does not crash. And *Quorum* $\geq n - \lceil n/4 \rceil - k \geq \lceil n/2 \rceil$ for $n \geq 4k+2$, that is, *Quorum* have majority processes. Therefore, each run R_i^1 is a legitimate run. We then construct run R_∞^1 based on the class of run R_i^1. The way of the construction and the definition of failure pattern for this run are the same as that in Lemma 3. Since $p_2 \notin A(t_i)$ for any i, we will not crash p_2 in R_∞^1. Using the same argument of Lemma 3, we can prove that the output of $\mathcal{D}$ in R_∞^1 satisfy the properties of Π_2^S, all correct processes take an infinite number of steps and all messages sent to the correct processes are eventually delivered. So R_∞^1 is a legitimate run. But, in run R_∞^1, no live-component exists in the output of algorithm T. Contradiction reached.

If in some finite run R_i^1, we have $p_2 \in A(t_i)$, we go to step 2 in which we construct another class of infinitely run to reach contradiction. Since $p_2 \in A(t_i)$, we have $p_2 \in$ a quasi-live component. Note that P_{11} crash in such run, so the output of algorithm T will satisfy $H'(p_2, t_i)$.*Quorum'* $\subsetneqq P_{12}$. In run R_1^2, we hold processes in P_{11} and do not let them take any steps until time t_i. Since the algorithm is deterministic, at time t_i, process and channel states are exactly like in run R_i^1. At time $s_1 = t_i + 1$, we crash all processes in P_{12}. We set *Quorum* $= P/P_{12}$, set *cid* to 0, and set *lbound* to 2. We choose one process p_3 in P_{11} and set its *isLeader* to *True*, and set *isLeader* outputs of all other processes to *False*. Then, we construct run R_j^2 using the same method in step 1. If at any time s_j we have $p_3 \notin A(s_j)$, we can also construct infinitely run class $\{R_j^2\}$ and then construct run R_∞^2. By the same argument in step 1, we can prove that run R_∞^2 is a legitimate run and no live-component exists in the output of algorithm T in such run. This contradictory.

Otherwise, there must exist j such that $p_3 \in A(s_j)$, so we have the output of algorithm T satisfy $H'(p_3, s_j)$.*Quorum*$' \subseteqq P_{11}$. Now we enter step 3, in which a run R is constructed as the following. The beginning of R works the same as run R_1^2 until time t_i. Then we hold processes in P_{12} and do not let them take any steps until time s_j. Since the algorithm is deterministic, at time s_j, process and channel states are exactly the same as in run R_j^2. Now we have $H'(p_3, s_j)$.*Quorum*$' \subseteqq P_{11}$ and $H'(p_2, t_i)$.*Quorum*$' \subseteqq P_{12}$. But both p_2 and p_3 are in component P_1 and $H(p_3, s_j)$.*Quorum*$' \cap H(p_2, t_i)$.*Quorum*$' = \emptyset$. Contradiction.

Therefore, in both cases we reach contradictions. The theorem thus holds. □

B Proof of Cornnectness of k-Set Agreement Algorithm

We first list a few straightforward propositions about operator $\cup_m$ and relation $\preceq_m$.

Proposition 5 *Given two sets R_1 and R_2 containing round numbers and a positive integer m, $R_1 \preceq_m R_2$ if and only if one of the following conditions hold: (a) $|R_2| < m$ and $R_1 \subseteq R_2$; or (b) $|R_2| = m$ and $\forall x \in R_1 \setminus R_2, \forall y \in R_2, x < y$.*

Proposition 6 *Given two sets R_1 and R_2 containing round numbers and a positive integer m, $R_1 \preceq_m R_2$ if and only if there exists a set R such that $R_1 \cup_m R = R_2$.*

Proposition 7 *Among all the sets of rounds with at most m elements, relation $\preceq_m$ is a partial order.*

All statements in the following proof refer to a run of the algorithm with a failure pattern F, a failure detector history H, a partition tree Γ and a partition split history S.

We start with some terminologies and notations used in the proof. When a proposer p is in its task 1, we say that it is in a *preparation phase* Φ^P if it is executing in lines 15–25; we say that it is in an *acceptance phase* Φ^A if it is executing in lines 26–35. Proposer p ends its preparation phase by either stopping its current task or entering its acceptance phase, and it ends its acceptance phase by either stopping its current task or deciding a value in line 35.

For any phase Φ (either a preparation phase or an acceptance phase) of a proposer p and any variable v of p, we denote $\Phi.v$ as the value of the variable v when p enters the phase (i.e., when p executes line 15 for the preparation phase or line 26 for the acceptance phase). We denote Φ^A.*node* as the node in the partition tree that corresponds to Φ^A.*p_cid* (the correspondence between *cid* outputs of the failure detector and the nodes in the partition tree is guaranteed by the property $(\Pi^S C1)$ of Π_k^S). We also use Φ^A.*prepQ* to represent Φ^A.*p_Quorum*.

An acceptance phase of p is successful if p decides at line 35 in the phase. In a successful acceptance phase Φ^A, we use Φ^A.*accQ* to represent the value of variable *p_Quorum* when p executes line 35. Informally, Φ^A.*prepQ* is the quorum of acceptors from which Φ^A.*est* is determined at the end of preceding preparation phase, and Φ^A.*accQ* is the quorum of acceptors by which p decides on the value Φ^A.*est* at the end of the acceptance phase.

For an acceptance phase Φ^A, we say that it belongs to the node Φ^A.*node* of the partition tree Γ in the run. A *path* of the partition tree Γ is a sequence of nodes from the root of Γ to a leaf node of Γ, where all nodes in the sequence have ancestor relationships with one another. We say that Φ^A belongs to a path of the partition tree if Φ^A.*node* is in the path.

Proposition 8 *If a proposer p enters an acceptance phase Φ^A and the preparation phase before Φ^A is Φ^P, then Φ^A.p_round $\in$ top(Φ^A.p_Rounds, Φ^P.lbound).*

Proof. Since p enters the acceptance phase Φ^A, we know that none of the conditions in line 22 are true. That is, all messages received from acceptors in Φ^A.*prepQ* are ACK-PREP messages with the same R value. According to line 45 on the acceptor thread, we know that Φ^A.*p_round* $\in$ *top*$(R, \Phi^P$.*lbound*). Therefore, by line 21, we have Φ^A.*p_Rounds* $= R$ and thus Φ^A.*p_round* $\in$ *top*(Φ^A.*pRounds*, Φ^P.*lbound*). □

Proposition 9 *Each acceptance phase Φ^A has a unique tuple (Φ^A.p_round, Φ^A.p_Rounds).*

Proof. For different proposer processes p and q, p is in acceptance phase Φ^A_p and q is in acceptance phase Φ^A_q. Then, since Φ^A_p.*p_round* $\equiv p(mod\, n)$ and Φ^A_q.*p_round* $\equiv q(mod\, n)$ by line 3 and line 11, so Φ^A_p.*p_round* $\neq \Phi^A_q$.*p_round*.

For a single proposer process p, suppose it is in an acceptance phase Φ^A. If it decides in line 35, it will not start task 1 again, that is, it will not be in another acceptance phase again. Otherwise, it must stop the task at line 30, line 32 or line 33. If it stops at line 30 or line 32, its *p_round* will increase in the next task. Otherwise, it must receive NACK-ACC from an acceptor q with $R \neq \perp$. According to line 50, we know Φ^A_p.*p_Rounds* $\preceq_n$ q.*a_Rounds*. And line 51 ensures Φ^A_p.*p_Rounds* $\neq$ q.*a_Rounds*. This directly leads to Φ^A_p.*p_Rounds* $\prec_n$ q.*a_Rounds* right before the execution of line 33. As a result, p.*p_Rounds* will change after line 33. Therefore, either *p_round* or *p_Rounds* increases when p stops task 1 without deciding. So, p cannot enter two different acceptance phases with the same tuple (Φ^A.*p_round*, Φ^A.*p_Rounds*). □

For any variable v of process p, we denote $v(p, t)$ as the value of v on p at time t after p executes its step at time t (if there is such a step).

Proposition 10 *On any acceptor q, the sequence of a_Rounds values ordered by time form a total order with respect to $\preceq_n$, that is, $t \leq t' \Rightarrow$ a_Rounds$(q, t) \preceq_n$ a_Rounds(q, t').*

Proof. This is guaranteed by the way *a_Rounds* variable is updated in lines 44 and 50, and by Proposition 6. □

Proposition 11 *For any acceptance phase Φ^A of a proposer p and any acceptor $q \in \Phi^A$.prepQ, let t be the time when q sends the* ACK-PREP *message to p, which puts q into Φ^A.prepQ before p enters phase Φ^A. Then we have $S(q, t) = \Phi^A$.node and a_Rounds$(q, t) = \Phi^A$.p_Rounds.*

Proof. Note that all received R's with $\Phi^A.taskid$ are the same (line 22) and *p_Rounds* before line 21 $\preceq_n$ *a_Rounds*(q, t) (by line 44). We then have *a_Rounds*$(q, t) = \Phi^A$.*p_Rounds* (by line 21 and the definition of $\preceq_n$). The fact that $S(q, t) = \Phi^A$.*node* is guaranteed by line 42, because q sends an ACK-PREP message, not a NACK-PREP message.[6] □

Proposition 12 *For any two acceptance phases Φ^A_1 and Φ^A_2 belonging to the same path in the partition tree, their p_Rounds are comparable, i.e., Φ^A_1.p_Rounds $\preceq_n$ Φ^A_2.p_Rounds or Φ^A_2.Rounds $\preceq_n$ Φ^A_1.p_Rounds. Moreover, Φ^A_2.node $\subsetneq$ Φ^A_1.node $\Rightarrow$ Φ^A_1.p_Rounds $\preceq_n$ Φ^A_2.p_Rounds and Φ^A_1.p_Rounds $\preceq_n$ Φ^A_2.p_Rounds $\Rightarrow$ Φ^A_2.node $\subseteq$ Φ^A_1.node.*

Proof. Since Φ^A_1 and Φ^A_2 belong to the same path $\mathcal{P}$, Φ^A_1.*prepQ* $\cap$ Φ^A_2.*prepQ* $\neq \emptyset$ by the property $(\Pi^S\Sigma 2)$ of failure detector Π^S_k. Suppose acceptor $q \in \Phi^A_1$.*prepQ* $\cap$ Φ^A_2.*prepQ*. By Proposition 11, there exists times t_1 and t_2 such that *a_Rounds*$(q, t_1) = \Phi^A_1$.*p_Rounds* and *a_Rounds*$(q, t_2) = \Phi^A_2$.*p_Rounds*. Then, by

[6]Here (and in a few other proofs) we implicitly rely on our model assumption that q executes lines 42–46 (and also lines 48–54) in one step at the same time. Such a reliance is for convenience and not essential. If q executes these lines at different times, it is easy to rephrase the statements so that the proofs are still correct.

Proposition 10 we know that *a_Rounds*(q, t_1) and *a_Rounds*(q, t_2) are comparable with respect to $\preceq_n$. Moreover, if Φ_2^A.*node* $\subsetneq \Phi_1^A$.*node*, by Proposition 11, it means $S(q, t_2) \subsetneq S(q, t_1)$, which implies $t_1 < t_2$. By Proposition 10, we have *a_Rounds*$(q, t_1) \preceq_n$ *a_Rounds*(q, t_2), and thus Φ_1^A.*p_Rounds* $\preceq_n \Phi_2^A$.*p_Rounds*. Reversely, if Φ_1^A.*p_Rounds* $\preceq_n \Phi_2^A$.*p_Rounds*, it means *a_Rounds*$(q, t_1) \preceq_n$ *a_Rounds*(q, t_2). By Proposition 10 we have $t_1 \leq t_2$, and thus $S(q, t_2) \subseteq S(q, t_1)$, which implies Φ_2^A.*node* $\subseteq \Phi_1^A$.*node*. □

Proposition 12 means that the order of the acceptance phases based on their *p_Rounds* values is compatible with the ancestor order of the corresponding nodes in the partition tree.

Lemma 13 *For all messages* (ACK-PREP, R, *TS*, v, $around$, *taskid*) *in* M_1 *computed in line 19, the TS values in these messages form a total order according to* $\preceq_n$.

Proof. We consider a preparation phase Φ^P of some proposer p, and let MSG1=(ACK-PREP,R,*TS*,v, $around$, *taskid*) and MSG2=(ACK-PREP,R',*TS*$'$,v',$around'$, *taskid*$'$) be two messages in M_1 of this phase. It is sufficient to prove *TS* and *TS*$'$ are comparable. Suppose the two messages are received from acceptors q and q' respectively. If one of the *TS* and *TS*$'$ is $\emptyset$, then the lemma holds, so we consider the case when both are not $\emptyset$.

Suppose q sends MSG1 at time t and q' sends MSG2 at time t'. Since both messages are ACK-PREP messages, we know that Φ^P.*p_cid* $\equiv$ *a_cid*(q, t) and Φ^P.*p_cid* $\equiv$ *a_cid*(q', t') by line 42. By the property $(\Pi^S C1)$ of Π_k^S, we know that $\equiv$ is an equivalence relation for any *cid* outputs appeared in one failure detector history H. So *a_cid*$(q, t) \equiv$ *a_cid*(q', t'). Again by property $(\Pi^S C1)$, we have $S(q, t) = S(q', t')$.

For any $t_1 \leq t$ and $t_2 \leq t'$, by the definition of partition tree and partition split history, we have $S(q, t) \subseteq S(q, t_1)$ and $S(q', t') \subseteq S(q', t_2)$. Thus we know that $S(q, t_1)$ and $S(q', t_2)$ are always on the same path in the partition tree Γ.

We choose time $t_1 \leq t$ to be the time when q updates its *a_TS* to value *TS* in line 53 (the time exists because *TS* is not $\emptyset$). This update is due to the receipt of an ACCEPT message from some proposer, so it corresponds to some acceptance phase Φ_1^A. According to line 48 and property $(\Pi^S C1)$, we know that Φ_1^A.*node* $= S(q, t_1)$. Similarly, we choose time $t_2 \leq t'$ to be the time when q' updates its *a_TS* to value *TS*$'$, and it corresponds to an acceptance phase Φ_2^A with Φ_2^A.*node* $= S(q', t_2)$. Therefore, we know that Φ_1^A and Φ_2^A belongs to the same path in the partition tree. By Proposition 12, we know that Φ_1^A.*p_Rounds* and Φ_2^A.*p_Rounds* are comparable w.r.t. $\preceq_n$. Finally from line 53, we know that q sets its *a_TS* to Φ_1^A.*p_Rounds* at time t_1 and q' sets its *a_TS* to Φ_2^A.*p_Rounds* at time t_2, so *TS* $= \Phi_1^A$.*p_Rounds* and *TS*$' = \Phi_2^A$.*p_Rounds*. Therefore, *TS* and *TS*$'$ are comparable w.r.t. $\preceq_n$. □

Lemma 13 guarantees that in line 25, it is always possible to find a timestamp *TS* with the highest value.

For each acceptance phase Φ^A, we define its predecessor, denoted as *pred*(Φ^A), to be another acceptance phase as the following. Before a proposer p enters its acceptance phase Φ^A, if it sets its *est* to its own proposal $proposal$ in line 24, then Φ^A has no predecessor, or *pred*$(\Phi^A) = \perp$. If p sets its *est* to a value v obtained from a message in M_1 in line 25, then there is an acceptor q which sends an ACK-PREP message with v and Φ^A.*taskid* to p. So there is an earlier time when q sets its *a_est* to v in line 53, which occurs when q receives an ACCEPT message with value v from a proposer p_0. Proposer p_0 must have sent this ACCEPT message in an acceptance phase Φ_0^A. In this case, we say that Φ_0^A is the predecessor of Φ^A, i.e., *pred*$(\Phi^A) = \Phi_0^A$. By definition, we have Φ^A.*est* = *pred*(Φ^A).*est* if *pred*$(\Phi^A) \neq \perp$. Informally, the predecessor list traces the acceptance phases back to its origin based on the chain of *est* values. We define *pred*$^0(\Phi^A) = \Phi^A$ and *pred*$^j(\Phi^A)$ = *pred*(*pred*$^{j-1}(\Phi^A)$) for $j \geq 1$.

Proposition 14 *Suppose there are two acceptance phases* Φ^A *and* Φ_0^A *such that pred*$(\Phi^A) = \Phi_0^A$. *Then* Φ^A *and* Φ_0^A *belong to the same path,*

Φ_0^A.p_Rounds $\preceq$ Φ^A.p_Rounds, and Φ^A.node $\subseteq$ Φ_0^A.node. Moreover, Φ_0^A.p_Rounds is the TS value associated with value v that proposer p selects in line 25 before it enters phase Φ^A.

Proof. Recall the definition of *pred*(Φ^A); that is, proposer p has phase Φ^A, proposer p_0 has phase Φ_0^A, and q is the acceptor that relates Φ^A with Φ_0^A. Suppose q sends its ACK-PREP message with value v to p as described in the definition at time t, and q sets its *a_est* to v at time $t_0 \leq t$. By Proposition 11, *a_Rounds*(q,t) = Φ^A.*p_Rounds* and Φ^A.*node* $= S(q,t)$. By Proposition 10, *a_Rounds*$(q,t_0) \preceq_n$ *a_Rounds*(q,t). By line 53, *a_Rounds*(q,t_0) = Φ_0^A.*p_Rounds*, and by line 48, Φ_0^A.*node* $= S(q,t_0)$. Therefore, Φ_0^A.*p_Rounds* $\preceq_n$ Φ^A.*p_Rounds*. By the definition of partition split history S, we have $S(q,t) \subseteq S(q,t_0)$. Thus Φ^A.*node* and Φ_0^A.*node* are on the same path and Φ^A.*node* $\subseteq \Phi_0^A$.*node*.

Let *TS* be the value associated with v that proposer p selects before it enters phase Φ^A. Just as in the definition of *pred*(Φ^A), we can follow the value v and find an acceptor q that sets its *a_est* to v and its *a_TS* to *TS* in line 53. According to line 53 and the definition of *pred*(Φ^A), *TS* is the same as *pred*(Φ^A).*p_Rounds*, which is Φ_0^A.*p_Rounds*. □

In the partition tree Γ, we call a node P_i a *critical node* if the following two conditions hold:

1. There exists a successful acceptance phase Φ^A with Φ^A.*node* $= P_i$;

2. There dose not exist any successful acceptance phase Φ^A with Φ^A.*node* $\supsetneqq P_i$. In other words, none of the ancestors of P_i satisfies condition 1.

For a critical node P_i, we define *TS*(P_i) to be the smallest Φ^A.*p_Rounds* (w.r.t. order $\preceq_n$) such that Φ^A.*node* $= P_i$ and Φ^A is a successful acceptance phase.

Proposition 15 *Let P_i and P_j be two different critical nodes, then $P_i \cap P_j = \emptyset$.*

Proof. If $P_i \cap P_j \neq \emptyset$, it must be true that P_i is P_j's ancestor or P_j is P_i's ancestor. According to condition 2, both cases are impossible. □

Proposition 16 *For any successful acceptance phase Φ^A, we can find an unique critical node P_i such that Φ^A.node $\subseteqq P_i$. We call cnode$(\Phi^A) = P_i$.*

Proof. Let $\Phi_0^A = \Phi^A$ and $P_{j_0} = \Phi_0^A$.*node*. If P_{j_0} is critical, the critical node is found. Otherwise, there must exists an acceptance phase Φ_1^A such that Φ_1^A.*node* $\supsetneqq P_{j_0}$. Therefore, $P_{j_1} = \Phi_1^A$.*node* is an ancestor of P_{j_0}. If P_{j_1} is critical, the critical node is found. Otherwise, we can find P_{j_2}, P_{j_3}, and so on in the similar way. Because nodes $P_{j_k} (k = 1, 2, \ldots)$ are distinct ancestors of P_{j_0} and the number of P_{j_0}'s ancestors are finite, there must be a node P_{j_k} which is a critical node. So there must exists at least one critical node P_i such that Φ^A.*node* $\subseteqq P_i$.

If there are two critical nodes P_i and P_j with Φ^A.*node* $\subseteqq P_i$ and Φ^A.*node* $\subseteqq P_j$. We have $\emptyset \neq \Phi^A$.*node* $\subseteqq P_i \cap P_j$. According to Proposition 15, this is impossible. Therefore, the critical node P_i is unique. □

For a component P_i in the partition tree Γ with a failure detector history H, recall that we define *lbound*$(P_i) = \max\{H(p,t).\textit{lbound} \mid p \in P_i$ and $t \in \mathcal{T}\}$.

Lemma 17 *For any successful acceptance phase Φ^A, there must be an acceptance phase Φ_0^A and some non-negative integer j such that Φ_0^A = pred$^j(\Phi^A)$, Φ_0^A.p_Rounds = TS(cnode(Φ^A)) and Φ_0^A.p_round $\in$ top(TS(cnode(Φ^A))), lbound(cnode(Φ^A))).*

Proof. Let p be the process that enters Φ^A, and the preparation phase p enters just before Φ^A is Φ^P.

If Φ^A.*p_Rounds* = *TS*(*cnode*(Φ^A)), by Proposition 8 we have Φ^A.*p_round* $\in$ *top*(*TS*(*cnode*(Φ^A))), Φ^P.*lbound*). Because Φ^A is a successful acceptance phase, it must be true that Φ^A.*node* $\subseteq$ *cnode*(Φ^A), which leads to $p \in$ *cnode*(Φ^A). According to the definition of *lbound*(*cnode*(Φ^A)), we know Φ^P.*lbound* $\leq$ *lbound*(*cnode*(Φ^A)). So Φ^A.*p_round* $\in$ *top*(*TS*(*cnode*(Φ^A))), *lbound*(*cnode*(Φ^A)))) and the lemma holds with Φ_0^A = *pred*$^0(\Phi^A)$.

Otherwise, we have *TS*(*cnode*(Φ^A)) $\preceq_n$ Φ^A.*p_Rounds*. Let Φ_u^A be an successful acceptance phase belonging to node $\mathit{cnode}(\Phi^A)$ with $\Phi^A_u.\mathit{p\_Rounds} = \mathit{TS}(\mathit{cnode}(\Phi^A))$, and p_u be the process that enters Φ^A_u. Since Φ^A and Φ^A_u are on the same path $\mathcal{P}$, by Proposition 12 $\Phi^A.\mathit{node} \subseteq \Phi^A_u.\mathit{node}$.

Let t be the time when p completes the repeat-until loop in lines 16–18 before it enters the acceptance phase Φ^A. Thus, the last *p_Quorum* value p reads is the failure detector output at time t, i.e., the value $H(p,t).\mathit{Quorum}$. By definition $\Phi^A.\mathit{prepQ} = H(p,t).\mathit{Quorum}$. Also, by the condition (3) in line 22, if at time t the *cid* output of the failure detector were not the same as *p_cid* obtained at the beginning of the preparation phase (w.r.t. relation $\equiv$), p would not enter the acceptance phase Φ^A. Therefore we have $S(p,t) = \Phi^A.\mathit{node}$ by the definition of $\Phi^A.\mathit{node}$ and the property $(\Pi^S C1)$ of Π^S_k.

Let t_u be the time when p_u completes the repeat-until loop in lines 27–29 in p_u's acceptance phase Φ^A_u. By a similar argument, we have $\Phi^A_u.\mathit{accQ} = H(p_u,t_u).\mathit{Quorum}$. Moreover, line 30 guarantees that $S(p_u,t_u) = \Phi^A_u.\mathit{node}$. Therefore, $S(p,t) \subseteq S(p_u,t_u)$.

Now we can apply property $(\Pi^S\Sigma 2)$ of Π^S_k and conclude that $\Phi^A.\mathit{prepQ} \cap \Phi^A_u.\mathit{accQ} \neq \emptyset$. Let q be the acceptor in $\Phi^A.\mathit{prepQ} \cap \Phi^A_u.\mathit{accQ}$. So at some time t_0 q sends message $(\text{ACK-ACC}, \mathit{taskid}')$ to p_u, and at another time t_1 q sends message $(\text{ACK-PREP}, \mathit{a\_Rounds}(q,t_1), \mathit{TS}, v, \mathit{taskid})$ to p. According to lines 53 and 51, we have $\mathit{a\_TS}(q,t_0) = \mathit{a\_Rounds}(q,t_0) = \Phi^A_u.\mathit{p\_Rounds}$. By Proposition 11, $\mathit{a\_Rounds}(q,t_1) = \Phi^A.\mathit{p\_Rounds}$.

Since $\Phi^A_u.\mathit{p\_Rounds} \preceq_n \Phi^A.\mathit{p\_Rounds}$ but they are not equal, we have $t_0 < t_1$. Thus $\mathit{a\_est}(q,t_0) \neq \perp$ implies that $\mathit{a\_est}(q,t_1) \neq \perp$, which means the value v in the message $(\text{ACK-PREP}, \Phi^A.\mathit{p\_Rounds}, \mathit{TS}, v, \mathit{taskid})$ q sends to p is not $\perp$. According to line 24, p cannot use its own proposal value in phase Φ^A. Therefore Φ^A has a predecessor.

Let $\Phi^A_1 = \mathit{pred}(\Phi^A)$. We now prove that $\Phi^A_u.\mathit{p\_Rounds} \preceq_n \Phi^A_1.\mathit{p\_Rounds}$. Since by lines 53 and 51 *a_TS* values of acceptor q always take *a_Rounds* values of q, by Proposition 10 *a_TS* values on q are also totally ordered with time. So we have $\mathit{a\_TS}(q,t_0) \preceq_n \mathit{a\_TS}(q,t_1)$. According to line 25, p will choose v with the highest timestamp *TS* before it enters Φ^A, so we have $\mathit{TS} \succeq_n \mathit{a\_TS}(q,t_1) \succeq_n \mathit{a\_TS}(q,t_0) = \Phi^A_u.\mathit{p\_Rounds}$. By Proposition 14, $\mathit{TS} = \mathit{pred}(\Phi^A).\mathit{p\_Rounds} = \Phi^A_1.\mathit{p\_Rounds}$. Therefore, we have $\Phi^A_1.\mathit{p\_Rounds} \succeq_n \Phi^A_u.\mathit{p\_Rounds}$. Again by Proposition 14, Φ^A_1 is also on path $\mathcal{P}$.

We can now repeat the above argument again on phase Φ^A_1. Note that we do not need the condition that Φ^A_1 is a successful acceptance phase, because we already have $\Phi^A_u.\mathit{p\_Rounds} \preceq_n \Phi^A_1.\mathit{p\_Rounds}$. Finally, by the definition of $\mathit{pred}(\Phi^A)$, each time we find a predecessor we must have followed the causal chain of messages backward in time. Therefore, the above argument cannot be repeated infinitely often, and it will stop at some acceptance phase Φ^A_v, and it must be the case that $\Phi^A_v.\mathit{p\_Rounds} = \mathit{TS}(\mathit{cnode}(\Phi^A))$.

Now we consider two phases Φ^A_0 and Φ^A_1 from the acceptance phase chain created in the above manner, such that $\mathit{pred}(\Phi^A_1) = \Phi^A_0)$, $\Phi^A_0.\mathit{p\_Rounds} = \mathit{TS}(\mathit{cnode}(\Phi^A))$, and $\Phi^A_1.\mathit{p\_Rounds} \succneqq \Phi^A_0.\mathit{p\_Rounds}$. This is possible because $\Phi^A.\mathit{p\_Rounds} \succneqq \mathit{TS}(\mathit{cnode}(\Phi^A))$ and $\Phi^A_v.\mathit{p\_Rounds} = \mathit{TS}(\mathit{cnode}(\Phi^A))$.

Since Φ^A_1 and Φ^A_u are both in path $\mathcal{P}$, we have $\Phi^A_1.\mathit{prepQ} \cap \Phi^A_u.\mathit{accQ} \neq \emptyset$. Let p_1 be the process which enters Φ^A_1 and q be an acceptor in $\Phi^A_1.\mathit{prepQ} \cap \Phi^A_u.\mathit{accQ}$. There exists a time t such that q sends ACK-ACC for Φ^A_u. The event that q sends ACK-PREP message to p_1 for Φ^A_1 must happen at a time $t' > t$, because $\Phi^A_1.\mathit{p\_Rounds} = \mathit{a\_Rounds}(q,t') \succneqq_n \mathit{a\_Rounds}(q,t) = \Phi^A_u.\mathit{p\_Rounds}$. Suppose the *a_est*, *a_TS*, and *a_round* in the message come from an acceptance phase Φ^A_x. It must be true that Φ^A_x writes its values on q at or after time t. Therefore, we have $\Phi^A_u.\mathit{p\_Rounds} \preceq_n \Phi^A_x.\mathit{p\_Rounds}$ and $\Phi^A_u.\mathit{node} \supseteqq \Phi^A_x.\mathit{node}$.

On the other hand, since p_1 picks the *a_est*, *a_TS*, and *a_round* values written by Φ^A_0. According to our algorithm, one of the following two conditions must be true: $\Phi^A_x.\mathit{p\_Rounds} \prec_n \Phi^A_0.\mathit{p\_Rounds}$; or $\Phi^A_x.\mathit{p\_Rounds} = \Phi^A_0.\mathit{p\_Rounds}$ and $\Phi^A_x.\mathit{p\_round} \leq \Phi^A_0.\mathit{p\_round}$.

$\Phi_u^A.\mathit{p\_Rounds} = \Phi_0^A.\mathit{p\_Rounds}$ and $\Phi_u^A.\mathit{p\_Rounds} \preceq_n \Phi_x^A.\mathit{p\_Rounds} \preceq_n \Phi_0^A.\mathit{p\_Rounds}$ leads to $\Phi_x^A.\mathit{p\_Rounds} = \Phi_0^A.\mathit{p\_Rounds}$. So we have $\Phi_x^A.\mathit{p\_round} \le \Phi_0^A.\mathit{p\_round}$. Because $\Phi_x^A.\mathit{node} \subseteqq \Phi_u^A.\mathit{node}$, we know $\Phi_x^A.\mathit{p\_round} \in \mathit{top}(\mathit{TS}(\mathit{cnode}(\Phi^A)), \mathit{lbound}(\mathit{cnode}(\Phi^A)))$. Then $\Phi_0^A.\mathit{p\_round} \in \mathit{top}(\mathit{TS}(\mathit{cnode}(\Phi^A)), \mathit{lbound}(\mathit{cnode}(\Phi^A)))$ can be derived from $\Phi_x^A.\mathit{p\_round} \le \Phi_0^A.\mathit{p\_round}$. □

Lemma 18 *For any critical node P_i, let $\mathcal{V}(P_i) = \{\Phi^A.est \mid \Phi^A$ is successful acceptance phase and $cnode(\Phi^A) = P_i\}$. Then $|\mathcal{V}(P_i)| \le lbound(P_i)$.*

Proof. According to Lemma 17, for any successful acceptance phase Φ_x^A with $\mathit{cnode}(\Phi^A) = P_i$, there exists an acceptance phase $\Phi_{x_0}^A$ and a non-negative integer j such that $\Phi_{x_0}^A = \mathit{pred}^j(\Phi_x^A)$, $\Phi_{x_0}^A.\mathit{p\_Rounds} = \mathit{TS}(P_i)$, and $\Phi_{x_0}^A.\mathit{p\_round} \in \mathit{top}(\mathit{TS}(P_i), \mathit{lbound}(P_i))$.

According to Proposition 9, every $\Phi_{x_0}^A$ has a unique tuple $(\Phi_{x_0}^A.\mathit{p\_Rounds}, \Phi_{x_0}^A.\mathit{p\_round})$. Because for all Φ_0^A $\Phi_0^A.\mathit{p\_Rounds} = \mathit{TS}(P_i)$ and $\Phi_0^A.\mathit{p\_round} \in \mathit{top}(\mathit{TS}(P_i), \mathit{lbound}(P_i))$, the set $\{\Phi_{x_0}^A\}$ contains at most $\mathit{lbound}(P_i)$ distinct elements. Since $\Phi_x^A.\mathit{est} = \Phi_{x_0}^A.\mathit{est}$, $|\mathcal{V}(P_i)| \le |\{\Phi_{x_0}^A\}| \le \mathit{lbound}(P_i)$. □

Lemma 19 (Uniform k-Agreement) *There are at most k different decision values.*

Proof. Let $\mathcal{C} = \{P_i | P_i \text{ is a critical node}\}$ and $L(\Gamma) = \{P | P \text{ is a leaf node in } \Gamma\}$.

Since only successful acceptance phase can decide values, by Lemma 18, we just need to prove the following

$$\sum_{P_i \in \mathcal{C}} \mathit{lbound}(P_i) \le k$$

By the definition of *lbound*, we have $\mathit{lbound}(P_i) = \max\{\mathit{lbound}(Q) | Q \subseteqq P_i \wedge Q \in L(\Gamma)\}$. So

$$\mathit{lbound}(P_i) \le \sum_{Q \in L(\Gamma) \wedge Q \subseteqq P_i} \mathit{lbound}(Q)$$

By Proposition 15, for any leaf Q, there are at most one critical node P_i satisfying $Q \subseteqq P_i$. So, we have

$$\sum_{P_i \in \mathcal{C}} \mathit{lbound}(P_i) \le \sum_{Q \in L(\Gamma)} \mathit{lbound}(Q) \le k$$

The last inequity is because of the specification of Π^S. □

Lemma 20 (Termination) *At least one proposer will eventually decide one value.*

Proof. It is sufficient to only consider the algorithm execution after a time t and the processes in a "live" component P_i. After time t, the output of Π_k^S on processes in component P_i satisfies the following conditions:

1. *lbound* of all proposers are the same and stable. We use lb to represent it.
2. *isLeader* is stable. We call the proposer with *isLeader* $=$ *True leader proposer*. Note that there is at least one leader proposer and at most lb leader proposers.
3. *Quorum* only contains correct acceptors.
4. *cid* and *a_cid* are stable on proposers and acceptors. That is, the partition will stop splitting eventually.

According to the definition Π_k^S, such component P_i and time t exists.

We first claim that there exists a time t_0, for all time $t_1, t_2 > t_0$ and for any proposer $p \in P_i$ and acceptor $q \in P_i$, $\mathit{p\_Rounds}(p, t_1) = \mathit{p\_Rounds}(p, t_2)$ and $\mathit{a\_Rounds}(q, t_1) = \mathit{a\_Rounds}(q, t_2)$.

Suppose, for a contradiction, that there exists some processes whose *p_Rounds* or *a_Rounds* changes infinitely often. This means at least one proposer p keeps increasing its *p_round* value. According to lines 8–11 in our algorithm, $p.\mathit{p\_round}$ increases if and only if 1) p is not decided yet; 2) $p.\mathit{isLeader}$ is *True*; and 3) $p.\mathit{incflag}$ $=$ *True* or $p.\mathit{p\_round} \notin \mathit{top}(p.\mathit{p\_Rounds}, p.\mathit{lbound})$. According to lines 30, 32, and 48, *incflag* is set to *True* only if $p.cid$ changes or on an acceptor q $q.a\_cid \not\equiv p.cid$. But after time t, $p.cid \equiv q.a\_cid \equiv P$ for any proposers p and acceptor q in P. Therefore, $p.isLeader$ eventually stays at the value of *False*. So $p.p\_round \notin top(p.p\_Rounds, p.lbound)$ must be *True* infinitely often. Let P_l be the set of proposers with *isLeader* set to *True* after t. Proposers in $P_i \setminus P_l$ will stop increase their *p_round* variables eventually, since only the leader proposers are allowed to increase their *p_round* variables. Let $P_{inc} \subseteq P_l$ be the set of processes that continuously increase their *p_round* value. There must be a time $t_0 > t$ such that $p\_round(p_1, t') > p\_round(p_2, t')$ for all $p_1 \in P_{inc}$, $p_2 \in P \setminus P_{inc}$, and $t' > t_0$. Because $|P_{inc}| \leq |P_l| \leq lb$ after t, we have $\{p\_round(p, t')|p \in P_{inc}\} \subseteq top(\{p\_round(p, t')|p \in P\}, lb)$. Now on every proposer $p \in P_{inc}$, $p.p\_round \notin top(p.p\_Rounds, p.lbound)$ becomes *False* because every *p_round* number in $p.p\_Rounds$ is smaller than or equal to its corresponding process's current *p_round* value. Therefore, no proposers can change their *p_round* variables at and after t_0. This contradicts to the direct result of the assumption that a process's *p_Rounds* or *a_Rounds* keep changing.

Suppose no proposer decides. Let proposer p be a leader proposer after t, p will repeatedly executes Task 1. Consider the first time p runs into Task 1 after time t_0. p must not chooses a new *p_round*, because the new *p_round* value will change the *p_Rounds* according to lines 10– 12.

Since *p_Quorum* contains only correct processes, so p will not be stuck at the waiting loops of lines 16–18 and lines 27–29. After t $p.cid$ stops change, so the third condition at line 22 and the condition at line 30 are unsatisfied. Also, because $p.cid \equiv q.a\_cid$ for all acceptor q after t, q will not send NACK-PREP and NACK-ACC at line 42 and line 48, respectively. Because of lines 21, 33, 44, and 50, $p.p\_Rounds = q.p\_Rounds$ for all acceptor q after time t_0. So the second condition at line 22 is unsatisfied. Also, acceptors will not send NACK-ACC messages at line 51. Moreover, since $p.p\_round \in top(p.p\_Rounds, lb)$ and $p.p\_Rounds = q.p\_Rounds$, acceptors will not send any NACK-PREP messages at line 45. Therefore, the first condition at line 22 and the condition at line 31 are not satisfied. As a consequence, p shall decide in line 35. This is contradictory. So the lemma is proved. □

Lemma 21 (Validity) *If a proposer decides v, then v has been proposed by some process.*

Proof. According to our algorithm, if Φ^A is a successful acceptance phase of proposer p, p decides $\Phi^A.est$. $\Phi^A.est$ either equals to $pred(\Phi^A).est$, or equals to $p.proposal$ when $pred(\Phi^A) = \bot$. If it is the latter case, the lemma holds. Otherwise, the same argument can be applied on $pred(\Phi^A)$ no matter whether it is a successful acceptance phase or not. Therefore, we can create a chain of acceptance phases. Since there are only a finite number of acceptance phases before Φ^A (w.r.t. $(\prec_n, \leq)$ on $(\Phi^A.p\_Rounds, \Phi^A.p\_round)$), the chain must stop on a phase Φ_0^A such that $pred(\Phi_0^A) = \bot$. And we know $\Phi_0^A.est = p_0.proposal$, given that the process enters Φ_0^A is p_0. Therefore, $\Phi^A.est = p_0.proposal$ and the lemma holds. □

Theorem 11 *The algorithm in Figures 2, 3 and 4 solves the k-set agreement problem with any failure detector in Π_k^S.*

Proof. The theorem follows directly from Lemma 19–21. □