(anti–Ωx $\times$ Σz)-based k-set Agreement Algorithms

Zohir Bouzid, Corentin Travers

$(\text{anti}-\Omega^x \times \Sigma_z)$-based k-set Agreement Algorithms

Zohir Bouzid*
University Pierre et Marie Curie - Paris 6, LIP6-CNRS 7606, France.
zohir.bouzid@lip6.fr

Corentin Travers†
LaBRI, Univ. of Bordeaux 1, France.
travers@labri.fr

September 20, 2010

Abstract

This paper considers the k-set agreement problem in a crash-prone asynchronous message passing system enriched with failure detectors. Two classes of failure detectors have been previously identified as necessary to solve asynchronous k-set agreement: the class anti-leader anti$-\Omega^k$ and the weak-quorum class Σ_k. The paper investigates the families of failure detector $(\text{anti}-\Omega^x)_{1\leq x\leq n}$ and $(\Sigma_z)_{1\leq z\leq n}$. It characterizes in an n processes system equipped with failure detectors anti$-\Omega^x$ and Σ_z for which values of k, x and z k-set-agreement can be solved. While doing so, the paper (1) disproves previous conjunctures about the weakest failure detector to solve k-set-agreement in the asynchronous message passing model and, (2) introduces the first indulgent algorithm that tolerates a majority of processes failures.

Keywords: Set-agreement, asynchrony, failure detectors, indulgent algorithms.

1 Introduction

The k-set-agreement problem *k-set-agreement* [9] is one of the fundamental problem in fault tolerant distributed computing. In this problem, n processes starting each with an initial private value are required to agree on at most k values chosen among their initial values. The problem generalizes the *consensus* problem, which corresponds to the case where $k = 1$. In an *asynchronous* system, it is well known that 1-set-agreement is impossible as soon as at least one process may fail by crashing [16], whereas the case $k = n$ does not require any coordination at all. For intermediate values of k $(1 < k < n)$, asynchronous k-set agreement tolerating t crash failures is possible if and only if $k > t$ [6, 24, 29].

Failure detectors A *failure detector* is a distributed oracle that provides processes with possibly unreliable information on failures [8]. According to the quality of the information, several classes of failure detectors can be defined. Starting with [26, 30], the failure detector approach has been

*Supported by the DIGITEO project PACTOLE and by the ANR projects SHAMAN and ALADDIN.
†On leave from University Pierre et Marie Curie - Paris 6. Supported by the ANR project SPREADS and by the INRIA project REGAL.

investigated to alleviate the k-set-agreement impossibility in asynchronous systems. An algorithm that tolerates unreliable failure detection is said to be *indulgent* towards its failure detector [18, 20]. Informally, an indulgent algorithm is always *safe*: it never violates the safety part of the problem it is supposed to solve, even when the underlying failure detector gives false information about failures.

The quest for the weakest failure detector for k-set-agreement Given a distributed problem P, a natural question is to determine the *weakest failure detector* for P, that is a failure detector D which is both *sufficient* to solve the problem – there is an asynchronous algorithm based on D that solves P – and *necessary*, in the sense that any failure detector D' that allows solving P can be used to emulate D.

The question of the weakest failure detector class for k-set agreement ($1 < k < n$) has been first stated in [28]. This line of research [10, 11, 19, 23] culminated with the work of Zieliński who established that the failure detector class anti$-\Omega^{n-1}$ is the weakest to solve $(n-1)$-set-agreement in the wait-free shared memory model [31]. This has later been generalized to any k, $1 \leq k < n$ by three independent groups [2, 14, 17]. Informally, a failure detector anti$-\Omega^k$ outputs sets of $n-k$ process ids such that some non faulty process id eventually never appear in the outputs.

The situation is different in the message passing model where the answer is known only for the two boundaries cases, i.e., $k = 1$ (consensus) and $k = n-1$ [13]. For consensus ($k = 1$), it has been shown that the class of *eventual leader* failure detector $\Omega =$ anti$-\Omega^1$ is the weakest failure detector in the asynchronous message passing model in which a majority of processes are non-faulty ($t < \frac{n}{2}$) [7]. This result is generalized to the *wait-free* environment in [12] where it is shown that $\Omega \times \Sigma$ is the weakest failure detector class for consensus when $t < n$. Intuitively, failure detector Σ provides a reliable quorum system: when queried, a failure detector of the class Σ returns a sets of processes ids, such that (1) any two sets intersect and (2) eventually, every set contains only ids of correct processes. Actually, Σ is the weakest failure detector to implement a register in the message passing model [5, 12].

Recently, the failure detector family $(\Sigma_k \times \Omega^k)_{1 \leq k < n}$ has been conjunctured to be the weakest failure detector classes for k-set-agreement [4]. Failure detector Σ_k and Ω^k generalizes the classes Σ and Ω respectively. Intuitively, a failure detector Σ_k allows up to k partitions: any collection of $k+1$ sets outputs by the failure detector contain at least two intersecting sets. Ω^k, which has been introduced by Neiger [27], outputs sets of k ids that eventually converge to a set including the id of a non-faulty process. It is shown in [4] that $\Sigma_{n-1} \times \Omega^{n-1}$ is equivalent to the loneliness failure detector $\mathcal{L}$ which is the weakest failure detector class for $(n-1)$-set-agreement [13]. Before this paper, nothing specific was known about the power of $\Sigma_x \times \Omega^x$ to solve k-set-agreement, for $1 < x < n-1$.

Content of the paper The paper investigates in the message passing model the computational power of the failure detector families $(\Sigma_x)_{1 \leq x \leq n}$ and $(\text{anti}-\Omega^z)_{1 \leq z \leq n}$ as far as k-set-agreement is concerned. Its main contributions are the following:

1. It has been shown that Σ_k is necessary to solve k-set-agreement, for each $k, 1 \leq k \leq n-1$ [4]. Moreover, for $k = 1$, $\Sigma_1 = \Sigma$ alone is not powerful enough to solve consensus whereas Σ_{n-1} is sufficient to solve $(n-1)$-set-agreement [4, 13]. We give necessary and sufficient conditions on the values of k, x and n in order to k-set-agreement to be solvable in an n processes message passing system enriched with Σ_x (Theorem 1, section 3). Roughly speaking, we show that

Σ_x allows to eliminate at most $\lfloor \frac{n}{x+1} \rfloor$ initial values, thereby generalizing prior results for the cases $k = 1$ [11] and $k = n - 1$ [13].

2. The paper then investigates the combined power of Σ_x and anti$-\Omega^z$. For $k \geq xz$, we present a k-set-agreement algorithm that tolerates any number of failures (Section 5).

   To ensure safety, namely that no more than x values are decided, we design a non-trivial generalization of the alpha abstraction which is at the core of indulgent consensus [21]. Our abstraction (called alpha_x, section 4) can be seen as an obstruction-free object that allows processes to store and retrieve at most x distinct values. Its implementation relies solely on a failure detector of the class Σ_x. Of note, as Σ_x can be simulated in an asynchronous message passing system when $t < \frac{xn}{x+1}$, we obtain a xz-set-agreement algorithm which is indulgent (towards the underlying failure detector of the class anti$-\Omega^z$) and tolerates $t < \frac{xn}{x+1}$ failures. To our knowledge, every prior indulgent algorithm assumes a majority of correct processes ($t < n/2$) or relies on a strong failure detector (e.g., Σ) that cannot be implemented in the asynchronous message passing model when a majority of processes may fail ($t \geq n/2$).

3. Finally, we show that for large enough values of n, there is no k-set-agreement algorithm based on $\Sigma_x \times \Omega^z$ if $k < xz$ (Theorem 2, section 5). This last result has two noteworthy corollaries. First, as anti$-\Omega^z$ can easily be simulated using the output of Ω^z, it implies that the previous algorithm is optimal. Second, it rules out $\Pi_k = \Sigma_k \times \Omega^k$ as a weakest failure candidate for k-set-agreement, thus disproving Bonnet and Raynal's conjuncture [4].

Roadmap The paper is made up of 6 sections. Section 2 describes the computing model and the families of failure detector we are interested in. Section 3 investigates the power of Σ_x with respect to the solvability of k-set agreement. The alpha_k abstraction is introduced in section 4, which presents also an Σ_k-based implementation. Section 5 then describes an indulgent k-set agreement algorithm that relies on the previous abstraction and a failure detector of the class anti$-\Omega^x$. A matching impossibility result is also presented. Finally, section 6 provides some concluding remarks.

2 System Model and Failures Detectors

Asynchronous message passing system with process crash failures The system consists in a set of n processes denoted $\Pi = \{p_1, \ldots, p_n\}$. Processes are asynchronous and may fail by crashing. Processes communicate via sending and receiving messages over an asynchronous network. Each pair of processes is connected by a bi-directional channel. The channels are asynchronous but reliable. Reliable means that there is no creation, alteration or loss of messages whereas asynchronous means that message transfer delays are finite but unbounded.

Processes may fail by *crashing*, i.e., prematurely stop executing their code. A process is *correct* in an execution if it never crashes in this execution; otherwise it is *faulty.* $t (1 \leq t < n)$ denotes an upper bound on the number of processes that can crash in a run. Given an execution, *Correct* denotes the set of correct processes.

Notation As in [25], $\mathcal{MP}_{n,t}$ denotes the asynchronous distributed system made of n processes, among which at most t may crash in any run. $\mathcal{MP}_{n,t}[X]$ denotes a system enriched with a failure detector of a class X.

The k-set agreement problem In the k-set agreement problem, each process proposes a value and has to decide a value such that the following properties are satisfied: *(Validity)* A decided value is a proposed value; *(Termination)* Every correct process eventually decides a value; *(Agreement)* The number of distinct decided values is at most k.

Families of failure detector classes For process p_i, FD_i^τ is the value output by the failure detector at time τ.

- *The eventual leader family* $(\Omega^k)_{1 \leq k \leq n}$. This family has been introduced in [27] to generalizes the class of failure detectors Ω defined in [7], with $\Omega^1 = \Omega$. A failure detector of the class Ω^k maintains at each process p_i a set of processes of size at most k (denoted LEADER_i) that satisfies the following property:
  - (Eventual multiple leadership). There is a time after which the sets LEADER_i contains forever the same set of processes and at least one process of this set is correct.
- *The quorum family* $(\Sigma_k)_{1 \leq k \leq n}$ [4]. A failure detector of the class Σ_k maintains at each process p_i a variable TRUSTED_i that contains a set of processes. The family generalizes the "quorum" failure detector $\Sigma = \Sigma_1$ introduced in [12]. The sets output by a failure detector of the class Σ_z satisfy:
  - (Completeness) There is a time after which every set TRUSTED_i contains only correct processes.
  - (Intersection) For every set $\mathcal{Q} = \{Q_1, \ldots, Q_{k+1}\}$ of $k+1$ sets output by the failure detector, there exists $Q_i, Q_j \in \mathcal{Q}, i \neq j$ such that $Q_i \cap Q_j \neq \emptyset$.

  Of note, a failure detector Σ_k can be implemented in $\mathcal{MP}_{n,t}$ provided that $\frac{kn}{k+1} > t$. To simulate a failure detector query, a process sends a REQUEST message to all processes and waits for matching RESPONSES. The set X made of the ids of the senders of the first $n-t$ responses received defines the result of the query. It is easy to see that completeness is ensured: eventually, only correct processes send responses. The intersection property follows from the fact that each simulated query returns a set of $n - t \geq \lfloor \frac{n}{k+1} \rfloor + 1$ identities. Hence, any collection of $k+1$ such sets contains at least two intersecting sets.
- *The anti-Ω family* $(\text{anti}-\Omega^k)_{1 \leq k \leq n}$ [31]. A failure detector of the class $\text{anti}-\Omega^k$ outputs at each process p_i a set ANTI-LEADER_i of $n-k$ processes ids. $\text{anti}-\Omega^1$ is equivalent to Ω. In every run, there is a correct process such that eventually each set output by the failure detector does not contain the identity of this process.
  - (Anti-leadership) $\exists p_c \in \mathit{Correct}, \exists \tau$ such that $\forall \tau' \geq \tau, \forall p_i \in \Pi, c \notin \text{ANTI-LEADER}_i^{\tau'}$.

3 Σ_z and k-set-agreement

Among other results, [11] shows that there is a k-set-agreement algorithm based on Σ_1 if $k > n/2$. On the other side ($k = n-1$), in [13] a $(n-1)$-set agreement message passing algorithm is presented. The algorithm relies on a failure detector called $\mathcal{L}$, which has been proved in [4] to be equivalent to Σ_{n-1}. Actually, it is also shown in [13] that failure detector $\mathcal{L}$ is the weakest failure detector for $(n-1)$-set-agreement in the wait-free message passing model ($t = n-1$). We generalize these boundary results to the entire family $(\Sigma_z)_{1\leq z\leq n}$. Specifically, we present a k-set-agreement algorithm based on Σ_z, provided that $k \geq n - \lfloor \frac{n}{z+1} \rfloor$. A simple matching impossibility result is also presented.

Theorem 1. *The k-set-agreement problem can be solved in $\mathcal{MP}_{n,n-1}[\Sigma_z]$ if and only if $k \geq n - \lfloor \frac{n}{z+1} \rfloor$*

Solving k-set-agreement with Σ_z The algorithm combines ideas borrowed from the $(n-1)$-set-agreement protocol based on failure detector $\mathcal{L}$ presented in [13] and a k-set-agreement protocol based on σ_{2k} [11]. In short, a failure detector of the class σ_{2k} provides the properties of the class Σ only to a subset of size $2k$ of the system. The algorithm is described in Figure 1.

Let $A_1, \ldots, A_{z+1}$ be a partition of the set of processes such that $\forall i, 1 \leq i \leq z, |A_i| = \lfloor \frac{n}{z+1} \rfloor$ and $|A_{z+1}| = \lfloor \frac{n}{z+1} \rfloor + (n \mod (z+1))$. Each process in set A_i tries to decide the proposal of some process that belongs to some partition $A_j, j < i$. To that end, each process $p \in A_i$ first sends its proposal to all processes in "higher" partitions, i.e., the processes that belong to the sets $A_{i+1}, \ldots, A_{z+1}$ (line 1). When a process receives a value w from a "lower" partition, it decides that value after broadcasting a DEC message carrying that value (line 5). A process that has not yet decided also decides w when it receives such a message $DEC(w)$ (Task T3). Note that the initial values of the processes in the "highest" partition (A_{z+1}) cannot be decided using this mechanism. Hence at most $n - |A_{z+1}| = z\lfloor \frac{n}{z+1} \rfloor$ are decided in that way.

The mechanism sketched above allows every correct process to eventually decide as soon as at least two partitions contain correct processes. However, it may happen that all correct processes are contained in a single partition A_i. We notice that in that case, the failure detector output at each process is eventually contained in A_i (by the completeness property of the class Σ_z). Henceforth, to prevent processes from waiting for values forever, each process p_i periodically checks its failure detector output; If the current set of trusted processes is contained in p_i's partition, p_i is allowed to decide its initial value (task T2, lines 6-8). The proof shows (Lemma 1) that the total number of decided values is at most $k = z\lfloor \frac{n}{z+1} \rfloor + (n \mod (z+1))$.

init $A_1, \ldots, A_{z+1}$ sets of processes such that $\forall i, j, i \neq j, A_i \cap A_j = \emptyset; \bigcup A_i = \Pi$;
$\forall i \in [1..z] |A_i| = \lfloor \frac{n}{z+1} \rfloor; |A_{z+1}| = \lfloor \frac{n}{z+1} \rfloor + n \mod (z+1)$

propose(v) % code for process $p \in A_i$
(1) **foreach** $q \in \bigcup_{j>i} A_j$ **do** *send* VAL(v) *to* q **endfor**
(2) start tasks T1, T2, T3

(3) Task T1: **when** VAL(w) is received **do**
(4) **foreach** $q \in \Pi$ **do** *send* DEC(w) *to* q **enddo**
(5) *decide* w; return

(6) Task T2: **repeat** $X \leftarrow \Sigma_z$-QUERY() **until** $X \subseteq A_i$
(7) **foreach** $q \in \Pi$ **do** *send* DEC(v) *to* q **enddo**
(8) *decide* v; return

(9) Task T3: **when** DEC(w) is received
(10) **foreach** $q \in \Pi$ **do** *send* DEC(w) *to* q **enddo**
(11) *decide*(w); return

Figure 1: k-set agreement algorithm in $\mathcal{MP}_{n,n-1}[\Sigma_z]$, $k = z\lfloor \frac{n}{z+1} \rfloor + (n \mod (z+1))$

Lemma 1. *The protocol described in the figure 1 solves k-set agreement in $\mathcal{MP}_{n,n-1}[\Sigma_z]$ for $k \geq n - \lfloor \frac{n}{z+1} \rfloor$*

Proof. Validity. This property follows from the fact that a process decides its own proposal (line 5) or the proposal of another process (line 8).

Termination. Let us assume for contradiction that a correct process never decides. A process decides when it receives a DEC($\cdot$) message, and a process always broadcasts a DEC($\cdot$) message before deciding. Hence, as links are reliable, no processes ever decide. Let $m = \max\{i : A_i \cap Correct \neq \emptyset\}$. We consider two cases:

- $\forall i \neq m, A_i \cap Correct = \emptyset$. Let p be a correct process that belongs to A_m. It follows from the completeness property of the class Σ_z that eventually each set X_p returned by the failure detector queries is such that $X_p \subseteq A_m$. Therefore, p eventually exits the repeat loop (line 6) and then decides (task T2, line 8): a contradiction.

- $\exists \ell \neq m : A_\ell \cap Correct \neq \emptyset$. By definition of m, $\ell < m$. Let $p \in A_\ell$ and $q \in A_m$ two correct processes. p sends a message VAL($\cdot$) which is eventually received by q (line 1). The first time q receives a message VAL($\cdot$), it sends a DEC($\cdot$) message and then decides (task T3): a contradiction.

Agreement: at most $k = z\lfloor \frac{n}{z+1} \rfloor + (n \mod (z+1))$ values are decided. To simplify the reasoning, we assume without loss of generality that any two proposed values are distinct[1]. For each $i, 1 \leq i \leq z+1$, let $\delta_i = |\{v : \exists p \in A_i \text{ s.t. } v \text{ is proposed by } p \wedge v \text{ is decided }\}|$. δ_i is the number of values that are decided among the values proposed by the processes in A_i. Let d the total number of decided values. Obviously $d = \sum_{i=1}^{z+1} \delta_i$.

Let us first observe (Observation *O1*) that there must exist a set A_j such that no processes $p \in A_j$ exit the repeat loop at line 6 (Task T2). In other words, any set X returned by a query performed by any process $p \in A_j$ is *not* included in A_j. Let us assume for contradiction that for each $A_i, 1 \leq i \leq z+1$, there exists a process $p_i \in A_i$ that gets back a set $X_i \subseteq A_i$ from an invocation of Σ_z-QUERY(). Since any two sets $A_i, A_j, i \neq j$ do not intersect, so do any two sets X_i, X_j. Hence the sets $X_1, \ldots, X_{z+1}$ are pairwise disjoint, contradicting the intersection property of the class Σ_z.

We associate to each decided value v the process $proc(v)$ that sends $DEC(v)$ for the first time (in tasks T1, line 4 or T2, line 7). Notice that (*O2*) if $v \neq v'$, $proc(v) \neq proc(v')$ since each process broadcasts $DEC(\cdot)$ at most once and, (*O3*) if v is proposed by some process that belongs to A_j then $proc(v) \in \bigcup_{i \geq j} A_j$. Moreover, if in addition A_j satisfies the observation *O1*, $proc(v) \in \bigcup_{i>j} A_i$. Hence, each initial value v of the processes in $A_j, A_{j+1}, \ldots, A_{z+1}$ that is decided has to be "relayed" by some process ($proc(v)$) that belongs to $\bigcup_{i>j} A_i$. By (*O2*), it then follows

$$\sum_{i=j}^{i=z+1} \delta_i \quad \leq \quad |A_{j+1}| + \ldots + |A_{z+1}| = (z-j+1)\lfloor \frac{n}{z+1} \rfloor + (n \mod (z+1))$$

By definition of δ_i, $\delta_i \leq |A_i|$. Hence,

$$\sum_{i=1}^{j-1} \delta_i \quad \leq \quad |A_1| + \ldots + |A_{j-1}| = (j-1)\lfloor \frac{n}{z+1} \rfloor$$

[1] If not, for the purpose of the analysis we think of the value proposed by p_i as a pair $\langle v, i \rangle$, for each process $p_i \in \Pi$.

By summing the two inequalities, we obtain

$$d = \sum_{i=1}^{z+1} \delta_i \leq (j-1)\lfloor\frac{n}{z+1}\rfloor + (z-j+1)\lfloor\frac{n}{z+1}\rfloor + (n \mod (z+1))$$
$$\leq z\lfloor\frac{n}{z+1}\rfloor + (n \mod (z+1)) = n - \lfloor\frac{n}{z+1}\rfloor \qquad \square$$

An impossibility result Together with Lemma 1, the following lemma completes the proof of Theorem 1.

Lemma 2. *$\forall n, k, z$ such that $k < n - \lfloor\frac{n}{z+1}\rfloor$, there is no k-set-agreement algorithm in $\mathcal{MP}_{n,n-1}[\Sigma_z]$*

Proof. Let us assume for contradiction that there exists a k-set-agreement algorithm $\mathcal{A}$ in $\mathcal{MP}_{n,n-1}[\Sigma_z]$, with $k < n - \lfloor\frac{n}{z+1}\rfloor$. We show that this implies that k-set-agreement can be solved in $\mathcal{MP}_{n,t}[\emptyset]$ for $t \geq k$, which is known to be impossible [6, 24, 29].

As noted earlier (section 2), Σ_z can be implemented in $\mathcal{MP}_{n,t}$ provided that $t \leq n - \lfloor\frac{n}{z+1}\rfloor - 1$. Algorithm $\mathcal{A}$ solves k-set-agreement for some $k \leq n - \lfloor\frac{n}{z+1}\rfloor - 1$ in $\mathcal{MP}_{n,n-1}[\Sigma_z]$. By combining algorithm $\mathcal{A}$ with a simulation of a failure detector Σ_z, we obtain a k-set-agreement algorithm in $\mathcal{MP}_{n,t}$, for $t = n - \lfloor\frac{n}{z+1}\rfloor - 1 \geq k$. This contradicts the fact that k-resilient k-set-agreement is impossible [6, 24, 29]. $\square$

4 The Alpha_k abstraction

This section presents the Alpha_k abstraction that generalizes the Alpha abstraction introduced by Guerraoui and Raynal in [21] to capture the safety part of indulgent consensus[2]. In the very same way, the abstraction Alpha_k captures the safety part of eventual failure detector based k-set-agreement algorithms. In short, the Alpha_k abstraction can be viewed as a shared object intended to store at most k values. A process accesses the object via the operation $\mathsf{propose}(\cdot)$ with as parameter a value it is willing to store and gets back one of the values actually stored in the object. However, in case of concurrent accesses, $\mathsf{propose}(\cdot)$ operations may not store any value and return the special value $\perp$, which is the object initial value.

More precisely, an alpha_k object exports one operation $\mathsf{propose}(v, r)$ with input parameters a value v and a round number r. As in [21], distinct processes must input distinct round numbers and each process must use strictly increasing round number. The Alpha_k abstraction is specified by the following properties, where $\perp$ is a special value that cannot be proposed:

- *Termination.* Every invocation of $\mathsf{propose}(\cdot)$ by any non-faulty process returns.
- *Validity.* If the invocation $\mathsf{propose}(v, r)$ returns $v' \neq \perp$, then $\mathsf{propose}(v', r')$ with $r' \leq r$ has been invoked.
- *k-Quasi-Agreement.* Let V be the set of non-$\perp$ values that are returned by $\mathsf{propose}(\cdot)$ invocations. $|V| \leq k$.
- *Conditional non-$\perp$convergence.* Let $I = \mathsf{propose}(\_, r)$ be a terminating invocation. If for every invocation $I' = \mathsf{propose}(\_, r')$ that starts before I returns, we have $r' < r$, I returns a non-$\perp$ value.

[2]Another generalization has been introduced in [28]. The implementation presented there relies on atomic registers which are not available in our settings.

4.1 Implementing Alpha_k with Σ_k

The algorithm implementing Alpha_k in an asynchronous message passing system is described in Figure 2. The algorithm relies on an underlying failure detector of the class Σ_k. It tolerates any number of failures.

Algorithm principles At any time, each process p_i has a value v (initially $\perp$) stored in the local variable val_i and a pair of integers $\langle r, \rho \rangle$ stored in the variables $\langle lre_i, pos_i \rangle$. The pair $\langle r, \rho \rangle$ can be seen as the *priority* of value v from p_i's point view. As in [21], lre stands for *last round entered.* r is the highest round number passed as a parameter of a $\mathsf{propose}(.)$ operation so far, as far as p_i knows. Furthermore, each round r is associated with a sequence of *positions* numbered from 1 to 2^r. When $\langle lre_i, pos_i, val_i \rangle = \langle r, \rho, v \rangle$, we say that *value v has reached position ρ in round r*. Also, based on its position ρ at round r, value v *logically occupies* a position ρ' at round $r + \delta$, for each $\delta > 0$. ρ' is defined by the following function g:

$$g(\rho, \delta) = 2^{\delta}(\rho - 1) + 1$$

Any pair of triplets $\langle r, \rho, v \rangle$, $\langle r', \rho', v' \rangle$, $r \leq r'$ can be compared via the function g: $\langle r, \rho, v \rangle \prec \langle r', \rho', v' \rangle$, i.e., v has a priority lower than v' iff $g(\rho, r' - r) < \rho'$[3].

An operation $\mathsf{propose}(v, r)$ returns a value $v' \neq \perp$ (possibly $v' \neq v$) only if v' has obtained a priority high enough so that no more that $k - 1$ values $\neq v'$ can be awarded higher priority. Operationally, a process p_i that invokes $\mathsf{propose}(v, r)$ proceeds as follows:

- In the first phase (lines 1-7), process p_i broadcasts the message REQ_R(r) in order (1) to inform other processes that it has entered round r and (2) to collect triplets $\langle round, position, value \rangle$ held by other processes.

  When a process p_j receives a message REQ_R(r), it first updates its round and the position of its value (using the function g) if $r > lre_j$. It then sends back the current value of its variables $\langle lre_i, pos_i, val_i \rangle$ in a response message RSP_R (lines 17-18).

  p_i is done collecting $\langle round, position, value \rangle$ triplets when it has received such values from each process p_j in a *quorum*, that is a set of processes returned by a query to the underlying failure detector Σ_k. If p discovers that another $\mathsf{propose}(\cdot)$ operation with input $r' > r$ has already started, it returns $\perp$ (line 5). Note that this does not violate the conditional convergence property. Otherwise, p_i selects among the values received the triplet with the highest priority, and updates its $\langle lre_i, pos_i, val_i \rangle$ accordingly (lines 6). In the case no triplets contain a value $\neq \perp$, p_i selects its own value with position 0 (line 7).

- The second phase (lines 8-16) consists in a repeat loop. In each iteration of the loop, p_i tries to increment the position of the value currently stored in val_i. To that end, it first broadcasts a request message REQ_W that carries p_i's current value together with its position and the current round r (lines 9).

  Process p_j that has learned that a round $> r$ has been started ignores the content of the messages REQ_W($\langle r, \rho, v \rangle$) it receives. Otherwise, p_j updates its round number and the position of its value. In addition, it adopts the received value if it has higher priority (lines

[3] When $g(\rho, r' - r) = \rho'$, v has a lower priority if $v < v'$. One can check that the $\prec$ relation is transitive, so g induces a total order on triplets $\langle r, \rho, v \rangle$.

19-24). Finally, p_j answers with a message RSP_W that carries the updated values of its variables $\langle lre_i, pos_i, val_i\rangle$ (lines 25).

As in the first phase, p_i stops collecting responses matching its request when a response message RSP_W$(\cdot)$ has been received from each process p_j in a quorum Q. Similarly, if one of the response carries a round number $> r$, p_i returns $\perp$. If this not the case, p_i adopts among the values received the triplet with the highest priority, and updates its $\langle lre_i, pos_i, val_i\rangle$ variables accordingly (lines 14). Since p_i always receives a response from itself, the value of pos_i at the end of the iteration is greater that the value of this variable at the end of the previous iteration. Finally, if the current value v of p_i reaches the last position associated with round r, v is returned (lines 15-16).

init $lre_i \leftarrow 0$; $val_i \leftarrow \perp$; $pos_i \leftarrow 0$;

function propose(r, v)
(1) **for each** $j \in \Pi$ *send* REQ_R(r) **end for**;
(2) **repeat** $Q \leftarrow \Sigma_k$-QUERY()
(3) **until** $(\forall p_j \in Q \cup \{p_i\}$: RSP_R$(r, \langle lre_j, pos_j, val_j\rangle))$ has been received from p_j)
(4) **let** $RCV = \{\langle lre_j, pos_j, val_j\rangle$: RSP_R$(r, \langle lre_j, pos_j, val_j\rangle)$ has been received $\}$;
(5) **if** $(\exists lre : \langle lre, \_, \_\rangle \in RCV\colon lre > lre_i)$ **then** *return* $(\perp)$ **endif**
(6) **let** $pos_M = \max\{pos : \langle r, pos, v\rangle\} \in RCV$;
$val_i \leftarrow \max\{v : \langle r, pos_M, v\rangle \in RCV\}$; $pos_i \leftarrow pos_M$;
(7) **if** $val_i = \perp$ **then** $val_i \leftarrow v$ **endif**
(8) **repeat** $pos_i \leftarrow pos_i + 1$;
(9) **for each** $p_j \in \Pi$ *send* REQ_W$(\langle r, pos_i, val_i\rangle)$ *to* p_j **end for**
(10) **repeat** $Q \leftarrow \Sigma_k$-QUERY()
(11) **until** $(\forall p_j \in Q \cup \{p_i\}$: RSP_W$(r, pos_i, \langle lre_j, pos_j, val_j\rangle))$ has been received from p_j)
(12) **let** $RCV = \{\langle lre_j, pos_j, val_j\rangle$: RSP_W$(r, pos_i\langle lre_j, pos_j, val_j\rangle)$ has been received $\}$;
(13) **if** $(\exists lre_j, \langle lre_j, \_, \_\rangle \in RCV : lre_j > r)$ **then** *return* $(\perp)$ **end if**
(14) **let** $pos_M = \max\{pos : \langle r, pos, v\rangle \in RCV\}$;
$val_i \leftarrow \max\{v : \langle r, pos_M, v\rangle \in RCV\}$; $pos_i \leftarrow pos_M$;
(15) **until** $(pos_i = 2^r)$
(16) *return* (val_i)

when REQ_R(rd) **is received from** p_j
(17) **if** $rd > lre_i$ **then** $pos_i \leftarrow g(pos_i, rd - lre_i)$; $lre_i \leftarrow rd$ **end if**
(18) *send* RESP_R$(rd, \langle lre_i, pos_i, val_i\rangle)$ *to* p_j

when REQ_W$(\langle rd, pos_j, val_j\rangle)$ **is received from** p_j
(19) **if** $(rd \geq lre_i)$ **then** $pos_i \leftarrow g(pos_i, rd - lre_i)$; $lre_i \leftarrow rd$
(20) **case** $pos_j > pos_i$ **then** $val_i \leftarrow val_j; pos_i \leftarrow pos_j$
(21) $pos_i = pos_j$ **then** $val_i \leftarrow \max(v_i, v_j)$
(22) $pos_j < pos_i$ **then** nop
(23) **end case**
(24) **end if**
(25) *send* RSP_W$(rd, pos_j, \langle lre_i, pos_i, val_i\rangle)$ *to* p_j

Figure 2: Implementing Alpha_k with Σ_k (code for p_i)

k-Quasi agreement The main difficulty is to guarantee that propose$(\cdot)$ invocations return collectively no more than k non-$\perp$ values. Value v_1 is returned at round r_1 if it reaches position $\rho_1 = 2^{r_1}$ and it has been adopted by a quorum Q_1. This means that for each process $q \in Q_1$, there is a point in time τ_q at which we have $\langle lre_q, pos_q, val_q\rangle = \langle r_1, \rho_1, v_1\rangle$. However, because quorums may not intersect, another value $v' \neq v_1$ may reach an arbitrary high position and consequently replaces the value v_1 at each process $q \in Q_1$. For example, this might happen if the quorums output by the failure detector during $\mathsf{propose}(r', \_)$ invocations with $r' > r_1$ do not intersect with Q_1. In these invocations, v' may be selected at the end of the first phase and its position can be increased in the second phase. In that case, v' has an higher priority than v_1, i.e., a process $q \in Q_1$ that receives $\langle r', \rho', v' \rangle$ will adopts v'.

The key idea of the algorithm is a as follows. Fix some round $r' > r_1$. In order to value v' to "overtake" value v_1 in round r', v' has to be adopted by a quorum Q' that does *not intersect* with Q_1. Consider the positions associated with round r'. At the beginning of round r', an odd position x might be logically occupied by a value v. This is the case if for some process p and some round $r < r'$, we have $\langle lre_p, pos_p, val_p \rangle = \langle r, \rho, v \rangle$ and $g(\rho, r' - r) = x$. Differently, by definition of g, each even position is initially free. Let x' and $x_1 = g(\rho_1, r' - r_1)$ be the positions logically occupied by values v' and v_1 respectively at the beginning of round r'. Observe that positions are increased by step of 1 and $x' + 2 \leq x_1$. So, to reach position x_1 value v' must first successfully go through position $x_1 - 1$. This can only happen if there is quorum Q' that adopts $\langle r', x_1 - 1, v' \rangle$. For each process $q \in Q_1$, the value v_1 held by q has an higher priority, since it logically occupies position x_1. So q cannot adopt $\langle r', x_1 - 1, v' \rangle$, hence $Q' \cap Q_1 \neq \emptyset$.

The rationale above can be extended to a chain of values $v_1, \ldots, v_\ell$ that each reaches higher and higher priorities to imply the existence of ℓ pairwise disjoint quorums. As any collection of $k+1$ quorums contains at least two intersecting quorums, the length of such a chain is at most k. In particular, this implies that at most k distinct values are returned – see the second part of the proof for more details (Lemmas 6–10).

Remark The algorithm is generic in the sense that the parameter k is never explicitly used in the code. In order to implement an $\mathsf{Alpha}_{k'}$ abstraction, it is sufficient to replace the underlying failure detector by a failure detector in the class $\Sigma_{k'}$. On the other hand, the algorithm uses 2^r positions per round. We have also developed along the same principles an algorithm that uses $O(r^{k-1})$ positions per round. However, determining which is the round r' position corresponding to a round $r < r'$ position, i.e., defining the equivalent of the g function, is more involved. As a result, the correctness proof is more intricate.

4.2 Proof

Consider a *well-formed* execution, in which processes execute the algorithm described in Figure 2 when $\mathsf{propose}(\cdot)$ is invoked. An execution is well-formed if the following conditions are fulfilled: (1) Only round number $r > 0$ are used as input parameters; (2) For any invocations $\mathsf{propose}(\_, r)$ and $\mathsf{propose}(\_, r')$ performed by processes p and p' respectively, if $p \neq p'$ then $r \neq r'$ and, if $p = p'$ and $\mathsf{propose}(\_, r)$ is invoked before $\mathsf{propose}(\_, r')$ then $r < r'$.

Lemma 3 (Termination)**.** *Every invocation of* $\mathsf{propose}(\cdot)$ *by a correct process terminates.*

Proof. Consider an invocation of $\mathsf{propose}(r, v)$ by a correct process p_i. The only possibility for the invocation to not terminate is that p_i blocks forever waiting for response messages corresponding to a read request (lines 2-3) or a write request (lines 10-11). We show that p_i cannot block while waiting for responses to a read request. This follows from (1) the fact that communication between correct processes are reliable, so every correct process eventually receives a message REQ_R(r), and then sends a RESP_R(r, _) message, which is eventually received by p_i, as well as (2) the completeness property of the class Σ_k which ensures that eventually every set Q returned by the failure detector contains only correct processes. The case of a write request is similar. ☐

Lemma 4 (Validity)**.** *Suppose that the invocation* $\mathsf{propose}(r, v)$ *returns* $v' \neq \perp$*. Then* $\mathsf{propose}(r', v')$ *with* $r' \leq r$ *has been invoked by some process.*

Proof. Consider an invocation $\mathsf{propose}(r, \_)$ by process p_i that returns a value $v' \neq \perp$. Observe that, for any process p_j, if the variables $\langle val_j, lre_j \rangle$ contain the values $\langle v', x \rangle$ then the value v' together with some round number $r' \leq x$ have been passed as parameters of a $\mathsf{propose}(\cdot)$ invocation. This follows from the facts that (1) the values of the variable lre_j are increasing (lines 17 and 19) and (2) a local variable lre can be set to round y only if $\mathsf{propose}(y, \_)$ has been invoked. When v' is returned, we have $val_i = v'$ and $lre_i = r$ (otherwise the invocation would have returned $\perp$) and validity thus follows. ☐

Lemma 5 (Conditional non-$\perp$convergence)**.** *Let* $I = \mathsf{propose}(r, \_)$ *be a terminating invocation. If for every invocation* $I' = \mathsf{propose}(r', \_)$ *that starts before* I *returns we have* $r' < r$*,* I *returns a non-*$\perp$ *value.*

Proof. Let $I = \mathsf{propose}(r, \_)$ be an invocation by process p_i that satisfies the condition of the lemma. The invocation $I = \mathsf{propose}(r, \_)$ can return $\perp$ only if a triplet $\langle r', \_, \_ \rangle$ is received with $r' > r$ (at lines 5 or 13). However, while I is being executed, the value of each variable lre_j is strictly smaller than r for every $p_j \neq p_i$. Hence, invocation I cannot return $\perp$. ☐

k-quasi agreement The next lemma is central in the proof of the k-quasi agreement property. Its proof is divided in three parts, namely, Lemma 8, 9 and 10. In the following, a quorum is a set of processes returned by a query to the underlying Σ_k failure detector.

Lemma 6. *Let* V *be the set non-*$\perp$ *values that are returned by the* $\mathsf{propose}(\cdot)$ *invocations.* $|V| = x \Rightarrow \exists x$ *quorums* $Q_1, \ldots, Q_x, \forall 1 \leq i < j \leq x, Q_i \cap Q_j = \emptyset$.

The k-quasi agreement property then follows easily from Lemma 6.

Lemma 7 (k-quasi agreement)**.** *Suppose that the protocol described in Figure 2 is instantiated with a failure detector of the class* Σ_k*. The total number of non-*$\perp$ *values that are returned by the* $\mathsf{propose}(\cdot)$ *invocations is at most* k*.*

Proof. Assume for contradiction that $x > k$ non-$\perp$ values are returned. It then follows from Lemma 6 that at least $k + 1$ disjoint quorums are output by the underlying failure detector Σ_k. This contradicts the intersection property of the class Σ_k. ☐

In order to prove Lemma 6, we define a sequence $S = s_1, \ldots, s_i = \langle r_i, \rho_i, v_i \rangle, \ldots$ where for each i, r_i is a round number, ρ_i a position associated to round r_i, and v_i a value. The sequence S is defined inductively as follows:

- r_1 is the smallest round r such that the invocation $\mathsf{propose}(\_, r)$ returns a non-$\perp$ value, if any. $\rho_1 = 2^{r_1}$ and v_1 is the value returned by that invocation.
- Suppose that $s_1, \ldots, s_{i-1}$ have been defined. r_i is the first round $r > r_{i-1}$ during which a value $v \neq \{v_1, \ldots, v_{i-1}\}$ reaches a position $\geq g(\rho_{i-1}, r - r_{i-1})$ (if such a round exists), i.e., $r_i = \min\{r : r > r_{i-1}, \exists p_x, \exists v \notin \{v_1, \ldots, v_{i-1}\}, \langle lre_x, pos_x, val_x \rangle = \langle r, g(\rho_{i-1}, r - r_{i-1}), v \rangle\}$. v_i is then this value, and we define $\rho_i = g(\rho_{i-1}, r_i - r_{i-1}) - 1$.

In the next lemma we give a formula for computing values ρ_i.

Lemma 8. *Suppose that $|S| \geq \ell$. $\forall i, 2 \leq i \leq \ell$, $\rho_i = 2^{r_i}(1 - \frac{1}{2^{r_1}} - \ldots - \frac{1}{2^{r_{i-1}}})$*

Proof. By definition, $\rho_2 = g(2^{r_1}, r_2 - r_1) - 1 = 2^{r_2 - r_1}(2^{r_1} - 1) + 1 - 1 = 2^{r_2}(1 - 1/2^{r_1})$. Suppose that $\rho_i = 2^{r_i}(1 - \frac{1}{2^{r_1}} - \ldots - \frac{1}{2^{r_{i-1}}})$.

$$\begin{aligned}
\rho_{i+1} &= g(\rho_i, r_{i+1} - r_i) - 1 \\
&= 2^{r_{i+1} - r_i}(2^{r_i}(1 - \frac{1}{2^{r_1}} - \ldots - \frac{1}{2^{r_{i-1}}}) - 1) + 1 - 1 \\
&= 2^{r_{i+1}}(1 - \frac{1}{2^{r_1}} - \ldots - \frac{1}{2^{r_{i-1}}}) - 2^{r_{i+1} - r_i} \\
&= 2^{r_{i+1}}(1 - \frac{1}{2^{r_1}} - \ldots - \frac{1}{2^{r_{i-1}}} - \frac{1}{2^{r_i}})
\end{aligned}$$

□

Suppose that value v reaches position ρ in round r, i.e., there exists a process p_i for which we have $\langle lre_i, pos_i, val_i \rangle = \langle r, \rho, v \rangle$ at some time. For every round $r' \geq r$, value v then logically occupies round r position $g(\rho, r' - r)$. Indeed, if process p_i later receives a read or write request carrying round $r' \geq r$, pos_i is updated to the value $g(\rho, r' - r)$ (at line 17 or line 19). Given a round r, we can then define the highest position logically occupied by value v as follows:

Definition 1. *Given a value v and a round number r, let* $\mathrm{mpos}(v, r)$ *denotes the maximal position logically occupied by value v at the beginning of round r. Formally,* $\mathrm{mpos}(v, r) = \max\{g(\rho', r - r') : \exists p_j, r' < r$ *and a time at which* $\langle lre_j, pos_j, val_j \rangle = \langle r', \rho', v \rangle\}$; *if no invocation* $\mathsf{propose}(r', v)$ *with* $r' < r$ *occurs,* $\mathrm{mpos}(v, r) = 0$.

Lemma 9. *Suppose that $|S| \geq \ell$. Let $\langle r, \rho, v \rangle$ be the value of process p_i variables $\langle lre_i, pos_i, val_i \rangle$ at some time. If $r \leq r_\ell$ and $v \notin \{v_1, \ldots, v_\ell\}$, $g(\rho, r_\ell - r) < \rho_\ell$.*

Proof. The proof is by induction on ℓ, the number of elements in the sequence S.

- $\ell = 1$. Let us first assume that $r < r_1$. The positions associated to round r are upper bounded by 2^r. Hence, $g(\rho, r_1 - r) = 2^{r_1 - r}(\rho - 1) + 1 \leq 2^{r_1 - r}(2^r - 1) + 1 = 2^{r_1} - 2^{r_1 - r} + 1 < 2^{r_1} = \rho_1$, as $r < r_1$.

  Let us assume now that $r = r_1$. Observe that in this case $g(\rho, r_1 - r) = \rho$. As the execution we consider is well-formed, there is a unique process, say p_1, that invokes PROPOSE(r_1, _). In round r_1, p_1 repeatedly picks the value with the higher position (from the point of view of round r_1, lines 6 and 14, the value and the position are then stored in variable val_1 and pos_1 respectively) and tries to increase this position by one (repeat loop, lines 8-15). p_1 is the only process doing so in round r_1. As by definition of r_1, the $\mathsf{propose}(r_1, \_)$ invocation returns v_1, p_1 successfully moves value v_1 from position $2^{r_1} - 1$ to position 2^{r_1}. By the first case, we know that $mpos(v, r_1) < 2^{r_1}$. But during round r_1, p_1 tries to move at most one value from position ϕ to position $\phi + 1$ for any position $\phi, 1 \leq \phi < 2^{r_1}$. Since $v \neq v_1$, the maximal position reached ρ by value v in round r_1 is thus $2^{r_1} - 1$.

- Suppose that the lemma is true for $\ell \geq 1$. If $r \leq r_\ell$, we have by induction hypothesis $g(\rho, r_\ell - r) + \delta \leq \rho_\ell$, for some $\delta \geq 1$. Per definition of g, $g(g(\rho, r_\ell - r) + \delta, r_{\ell+1} - r_\ell) \leq g(\rho_\ell, r_{\ell+1} - r_\ell)$, from which we have $2^{r_{\ell+1} - r_\ell}(g(\rho, r_\ell - r) + \delta - 1) + 1 \leq g(\rho_\ell, r_{\ell+1} - r_\ell)$. It thus follows $g(g(\rho, r_\ell - r), r_{\ell+1} - r_\ell) + 2^{r_{\ell+1} - r_\ell}\delta \leq g(\rho_\ell, r_{\ell+1} - r_\ell)$. Recall that $r_\ell < r_{\ell+1}$ and $\rho_{\ell+1} = g(\rho_\ell, r_{\ell+1} - r_\ell) - 1$. Moreover, $g(g(\alpha), \delta_1), \delta_2) = g(\alpha, \delta_1 + \delta_2)$ for any $\alpha, \delta_1, \delta_2$. We thus obtain $g(\rho, r_{\ell+1} - r) < \rho_{\ell+1}$ ($\star$).

Let us now assume that $r_\ell < r < r_{\ell+1}$. As $v \notin \{v_1, \ldots, v_\ell\}$, we must have by definition of $s_{\ell+1}$ that $\rho < g(\rho_\ell, r - r_\ell)$. As $r < r_{\ell+1}$, we apply the same reasoning as the previous case to get $g(\rho, r_{\ell+1} - r) < \rho_{\ell+1}$ $(\star\star)$.

For $r = r_{\ell+1}$, observe that it follows from $(\star)$ and $(\star\star)$ that the maximal position logically occupied by value v at the beginning of round $r_{\ell+1}$ is $mpos(v, r_{\ell+1}) \leq \rho_{\ell+1} - 1$. Note that the reasoning is still valid for the value $v_{\ell+1}$, i.e., $mpos(v_{\ell+1}, r_{\ell+1}) \leq \rho_{\ell+1} - 1$. As (1) value $v_{\ell+1}$ reaches position $\rho_{\ell+1}$ (2) only one process, say $p_{\ell+1}$ increases positions in round $r_{\ell+1}$ and (3) $p_{\ell+1}$ tries to move at most one value from position ϕ to position $\phi + 1$, for every position ϕ, it follows that value $v \neq v_{\ell+1}$ cannot reach any position $\geq \rho_{\ell+1}$ in round $r_{\ell+1}$. □

Lemma 10. *Let V be the set non-$\perp$ values that are returned by the* propose$(\cdot)$ *invocations. If $|V| = x$, $s_1, \ldots, s_x$ are well defined.*

Proof. We inductively check that $s_1, \ldots, s_x$ are well defined.

- s_1. As $V \neq \emptyset$, at least one invocation of propose$(\cdot)$ returns a non-$\perp$ value. Hence, $r_1 = \min\{r :$ propose$(r, _)$ returns a non-$\perp$ value $\}$ is well defined. Moreover, as any two invocations have distinct round numbers as input parameters, there is an unique invocation I_1 (by process p_1) with input round number r_1. Thus the value v_1 returned by this invocation is well defined.

- $s_{\ell+1}, \ell < x$. Suppose that $s_1, \ldots, s_\ell$ are defined. As $|V| = x \geq \ell + 1$, there exists a value $v \notin \{v_1, \ldots, v_\ell\}$ that is returned by a propose$(\cdot)$ invocation. Let r be the round number used as input in this invocation. v reaches the final position $\rho = 2^r$ of that round (lines 15-16).

  Note that by definition of s_1, we have $r > r_1$. Assume for contradiction that $r \leq r_\ell$. Let $m, 1 < m \leq \ell$ such that $r_{m-1} < r \leq r_m$. We have, according to the definition of g and per Lemma 8:

$$\begin{aligned} g(\rho, r_m - r) &= 2^{r_m - r}(2^r - 1) + 1 \\ &= 2^{r_m}(1 - \frac{1}{2^r}) + 1 \\ \rho_m &= 2^{r_m}(1 - \sum_{j=1}^{m-1} \frac{1}{2^{r_j}}) \end{aligned}$$

  As $\forall j, 1 \leq j \leq m-1, r_j < r$, we get:

$$\rho_m = 2^{r_m}(1 - \sum_{j=1}^{m-1} \frac{1}{2^{r_j}}) < 2^{r_m}(1 - \frac{1}{2^r}) + 1 = g(\rho, r_m - r)$$

  which contradict lemma 9 because $r \leq r_m$. Hence, $r_\ell < r$. Finally, observe that (Lemma 8)

$$g(\rho_\ell, r - r_\ell) = 2^r(1 - \sum_{j=1}^{\ell} \frac{1}{2^{r_j}}) < 2^r = \rho$$

  Consequently, $r(v) > r_\ell$, the first round in which v reaches a position $\geq g(\rho_\ell, r(v) - r_\ell)$ is well defined. Therefore $r_{\ell+1} = \min\{r(v) : v \in V \backslash \{v_1, \ldots, v_\ell\}\}$ is well defined. □

We are now ready to prove Lemma 6. To do so we associate to each $s_i \in S$ a quorum Q_i. Intuitively, the processes in Q_i are those processes that allow value v_i to reach position ρ_i during round r_i. Each process $q \in Q_i$ hence holds the triplet $\langle r_i, \rho_i, v_i \rangle$ at some time. Note that, after that time, the round r and position ρ are always such that $r \geq r_i$ and $\rho \geq g(\rho_i, r - r_i)$. The crucial observation is that q cannot allow any value $v_j \neq v_i$ to reach position ρ_j, essentially because either $r_i > r_j$ (in the case $i > j$) or $g(\rho_i, r_j - r_i) > \rho_j$ (if $j > i$).

Proof of Lemma 6. Suppose that $|V| = x$. Let $\ell, 1 \leq \ell \leq x$. We first bound $mpos(v_\ell, r_\ell)$. Suppose that $\langle r, \rho, v_\ell \rangle$ are stored by some process p, with $r \leq r_\ell$. There are two cases:

- $1 \leq r \leq r_{\ell-1}$. Since $v_\ell \notin \{v_1, \ldots, v_{\ell-1}\}$, it follows from Lemma 9 that $g(\rho, r_{\ell-1} - r) < \rho_{\ell-1}$. Hence, $g(g(\rho, r_{\ell-1} - r), r_\ell - r_{\ell-1}) < g(\rho_{\ell-1}, r_\ell - r_{\ell-1})$ from which we have $g(\rho, r_\ell - r) < g(\rho_{\ell-1}, r_\ell - r_{\ell-1})$.
- $r_{\ell-1} < r < r_\ell$. By definition of s_ℓ, we have $\rho < g(\rho_{\ell-1}, r - r_{\ell-1})$. Therefore $g(\rho, r_\ell - r) < g(g(\rho_{\ell-1}, r - r_{\ell-1}), r_\ell - r)$ which implies $g(\rho, r_\ell - r) < g(\rho_{\ell-1}, r_\ell - r_{\ell-1})$.

We conclude that $mpos(v_\ell, r_\ell) < g(\rho_{\ell-1}, r_\ell - r_{\ell-1}) = \rho_\ell + 1$. By definition of $g(\cdot)$, $\rho + 1$ is odd. Similarly, as there exists $r' < r_\ell, \rho'$ such that $mpos(v_\ell, r_\ell) = g(\rho', r_\ell - r')$, $mpos(v_\ell, r_\ell)$ is odd. Consequently $mpos(v_\ell, r_\ell) < \rho_\ell$.

We now define a quorum Q_ℓ associated with the triplet $\langle r_\ell, \rho_\ell, v_\ell \rangle$. By definition of s_ℓ, r_ℓ is the first round during which value v_ℓ reaches a position $\geq \rho_\ell + 1$. There is a (unique) process p_ℓ that invokes $\mathsf{propose}(\cdot)$ with input parameter r_ℓ. Otherwise, round r_ℓ is never entered and value v_ℓ cannot reach position $g(\rho_{\ell-1}, r_\ell - r_{\ell-1}) = \rho_\ell + 1$ in round r_ℓ.

Note that (1) value v_ℓ reaches a position $\geq \rho_\ell + 1$ in round r_ℓ, (2) the highest position logically occupied by v_ℓ at the beginning of round r_ℓ is $< \rho_\ell$. Moreover, (3) only process p_ℓ increases positions in round r_ℓ, and (4) p_ℓ tries to move at most one value from position ϕ to position $\phi + 1$, for every position ϕ. It then follows that p_ℓ successfully moves value v_ℓ from position $\rho_\ell - 1$ to position $\rho_\ell + 1$. In more details, this means that the variable pos_ℓ successively contains the values $\rho_\ell - 1, \rho_\ell, \rho_\ell + 1$ while the variables $\langle lre_\ell, val_\ell \rangle$ keep the values $\langle r_\ell, v_\ell \rangle$.

In particular, let us consider the iteration of the repeat loop (lines 8-15) in which $pos_\ell = \rho_\ell$. Let Q_ℓ be the quorum that allows the inner repeat loop to terminate (lines 10-11). Observe that Q_ℓ is a set of process returned by a query to failure detector Σ_k. For each $q \in Q_\ell$, the message RSP_W received from q must carry the triplet $\langle r_\ell, \rho_\ell, v_\ell \rangle$. If not, p_ℓ either picks another pair $\langle \rho, v \rangle$ with $v \neq v_\ell$ and $\rho \geq \rho_\ell$ or returns $\perp$. In both case, p stops moving value v_ℓ. It cannot move v_ℓ later in the same round, as the highest position occupied by v_ℓ is ρ_ℓ, and in subsequent iterations, only values located at position $> \rho_\ell$ can be moved.

Consequently, it follows that $\forall p_i \in Q_\ell$ there exists a time τ_i^ℓ at which we have $\langle lre_i, pos_i, val_i \rangle = \langle r_\ell, \rho_\ell, v_\ell \rangle$.

Finally, we establish that $\forall i, j, 1 \leq i < j \leq \ell$, $Q_i \cap Q_j$. Observe that if $\langle r_1, \rho_1, v_1 \rangle$ and $\langle r_2, \rho_2, v_2 \rangle$ are the values of the same process variables $\langle lre, pos, val \rangle$ at times $\tau_1 < \tau_2$ respectively, $(r_1 = r_2 \wedge \rho_1 \leq \rho_2) \vee (r_1 < r_2 \wedge g(\rho_1, r_2 - r1) \leq \rho_2)$ $(\star\star\star)$.

Assume for contradiction that $\exists \ell, m, 1 \leq \ell < m \leq x$ such that $Q_\ell \cap Q_m \neq \emptyset$. Let $p_i \in Q_\ell \cap Q_m$. There are two cases:

- $\tau_i^\ell < \tau_i^m$. In that case, p_ℓ sends first a message RSP_W carrying $\langle r_\ell, \rho_\ell, v_\ell \rangle$ and later a message RSP_W carrying $\langle r_m, \rho_m, v_m \rangle$. Note that the two triplets are the values at times τ_i^ℓ and τ_i^m

respectively of the variables $\langle lre_i, pos_i, val_i \rangle$. We have:

$$g(\rho_\ell, r_m - r_\ell) = 2^{r_m}(1 - \sum_{j=1}^{\ell} \frac{1}{2^{r_j}}) + 1 \quad \text{and } \rho_m = 2^{r_m}(1 - \sum_{j=1}^{m-1} \frac{1}{2^{r_j}})$$

from which we obtain $\rho_m < g(\rho_\ell, r_m - r_\ell)$, contradicting observation $(\star\star\star)$.

- $\tau_i^\ell > \tau_i^m$. This implies that lre_i first contains r_m and later $r_\ell < r_m$, which is impossible according to observation $(\star\star\star)$. $\square$

5 A k-set agreement algorithm

This section presents an (anti$-\Omega^x \times \Sigma_z$)-based k-set-agreement protocol, and a matching impossibility result on solving k-set agreement in the family of systems $(\mathcal{MP}_{n,n-1}[\text{anti}-\Omega^x, \Sigma_z])_{1 \le x,z \le n}$. The main results of this section are summarized by the following theorem:

Theorem 2. *The k-set-agreement problem can be solved in $\mathcal{MP}_{n,n-1}[\text{anti}-\Omega^x, \Sigma_z]$ if $k \geq xz$. Moreover, if $2xz \leq n$, the k-set-agreement problem cannot be solved in $\mathcal{MP}_{n,n-1}[\Omega^x, \Sigma_z]$ if $k < xz$.*

5.1 Solving k-set agreement with anti$-\Omega^x$ and Σ_z

For the system $\mathcal{MP}_{n,n-1}[\text{anti}-\Omega^x, \Sigma_z]$, we describe a k-set agreement algorithm that requires $k \geq xz$. From a computability point of view, our algorithm is optimal if n is large enough: we later establish that if $k < xz$ and $n \geq 2xz$ there is no k-set agreement algorithm in $\mathcal{MP}_{n,n-1}[\text{anti}-\Omega^x, \Sigma_z]$ (Corollary 1).

Using Ω and Σ_z to solve k-set agreement for $k \geq z$ The algorithm is a simple adaptation of the generic Ω-based consensus algorithm presented in [21], in which an Alpha_k object is used in place of an Alpha object. For completeness, the algorithm is described in Figure 3. The fact that any decided value has been returned by an invocation of $\mathsf{Alpha}_k.\mathsf{propose}(\cdot)$ guarantees validity and agreement. Because eventually a unique correct process considers itself the leader, there is a time after which only this process invokes $\mathsf{Alpha}_k.\mathsf{propose}(\cdot)$. Hence, by the conditional convergence property of the object, there is an invocation that returns a non-$\perp$ value. This value is then broadcast, allowing every non-faulty process to decide, therefore ensuring termination.

```
SA_propose(v)
   dec_i ← ⊥; r_i ← i;
   while (dec_i = ⊥) do
      if Ω-QUERY() = i then dec_i ← Alpha_k.propose(r_i, v)
                                  r_i ← r_i + n end if end do
   for each p_j ∈ Π do send DECIDE(dec_i) to p_j end do

when DECIDE(w) is received do
   for each p_j ∈ Π do send DEC(w) to p_j end do
   decide w; return
```

Figure 3: k-set agreement algorithm in $\mathcal{MP}_{n,n-1}[\Omega, \Sigma_k]$, code for p_i

Using anti$-\Omega^x$ **and** Σ_z **to solve** k**-set agreement for** $k \geq xz$ Our algorithm is based on a failure detector vector$-\Omega^x$ [31]. A failure detector of the class vector$-\Omega^x$ is a vector of x sub-detectors, $\Omega_1, \ldots, \Omega_x$, such that at least one Ω_i is a failure detector of the class Ω. When $k = n-1$, the vector$-\Omega$ failure detector proposed in [31] is obtained. It was shown there how vector$-\Omega$ can be implemented from anti$-\Omega^{n-1}$ in the wait-free asynchronous shared memory model, and how it can be used to solve $(n-1)$-set agreement. The failure detector vector$-\Omega^x$ was also presented in [31]. It is claimed there that the algorithm to transform anti$-\Omega^{n-1}$ into vector$-\Omega^{n-1}$ (Figure 1 in [31], see also [3]), can be generalized to transform anti$-\Omega^x$ into vector$-\Omega^x$. A close look at the transformation algorithm reveals that it can be easily adapted to the message passing case if a reliable broadcast primitive is available. As reliable broadcast can be implemented in an asynchronous message passing system in which any number of processes may fail [22], vector$-\Omega^x$ can be implemented in $\mathcal{MP}_{n,n-1}[\text{anti}-\Omega^x]$.

To solve k-set agreement in $\mathcal{MP}_{n,n-1}[\text{anti}-\Omega^x, \Sigma_z]$, processes simulate outputs of a failure detector vector$-\Omega^x$. We associate to each sub-detector $\Omega_i, 1 \leq i \leq x$ an instance of the (Ω, Σ_z)-based z-set agreement algorithm described in Figure 3. Each processes participates simultaneously in each of the x instances, and terminates as soon as it decides in one instance.

It follows from the fact that at least one sub detector Ω_i is a failure detector of the class Ω that at least one instance terminates. Moreover, since at most z values are decided in each instance, the total number of decided value is upper bounded by xz. Therefore,

Lemma 11. *Let $1 \leq k, x, z \leq n$. There is a k-set agreement algorithm in $\mathcal{MP}_{n,n-1}[\text{anti}-\Omega^x, \Sigma_z]$ if $k \geq xz$.*

5.2 An impossibility result

This section investigates k-set-agreement solvability when the system is enriched with failure detectors of both classes Ω^y and Σ_z. The main result is Lemma 13 which establishes that there is no k-set agreement algorithm in the wait-free environment $(t = n-1)$ where failure detectors Ω^y and Σ_z are provided if $k < yz$.

Lemma 12. *Let $k, 1 \leq k \leq n$ and $x, 1 \leq 2x \leq n$. If $k < x$, there is no k-set agreement algorithm in $\mathcal{MP}_{n,n-1}[\Omega^x, \Sigma]$.*

Proof. Assume for contradiction that there exists an algorithm $\mathcal{A}$ that solves k-set agreement in system $\mathcal{MP}_{n,n-1}[\Omega^x, \Sigma]$ for $k < x \leq n - \lfloor \frac{n}{2} \rfloor$. As failure detector Σ can be implemented in $\mathcal{MP}_{n,t}$ if $t < \frac{n}{2}$, it follows that one can combine an emulation of Σ and algorithm $\mathcal{A}$ to solve the k-set agreement problem in $\mathcal{MP}_{n,t<\frac{n}{2}}[\Omega^x]$ with $k < x$, which is not possible (see Theorem 4 in [25]). $\square$

Lemma 13. *Let $k, 1 \leq k \leq$ and $x, z, 1 \leq 2xz \leq n$. If $k < xz$, there is no k-set agreement algorithm in $\mathcal{MP}_{n,n-1}[\Omega^x, \Sigma_z]$.*

Proof. The proof relies on a partition argument. Let $\mathcal{A}$ be a k-set agreement algorithm in system $\mathcal{MP}_{n,n-1}[\Omega^x, \Sigma_z]$. We show that $k \geq xz$. Let $A_1, \ldots, A_z$ be a partition of Π such that $\forall i, 1 \leq i \leq z, 2x \leq |A_i|$. As $2xz \leq n$, there is such a partition.

For each $i, 1 \leq i \leq z$, let $\mathcal{E}_i$ be the set of executions of algorithm $\mathcal{A}$ that satisfy the following properties: (1) Each process that belongs to $\Pi \backslash A_i$ fails before taking any step; (2) Any two sets Q, Q' output by the failure detector Σ_z at any processes $p, p' \in A_i$ have a non-empty intersection; (3) Any set L output by the failure detector Ω^x at any process $p \in A_i$ is contained in A_i.

We claim that there exists an execution $e_i \in \mathcal{E}_i$ in which at least x distinct values are decided. Assume for contradiction that in every execution $e \in \mathcal{E}_i$ strictly less than x values are decided. According to the second and the third properties, from the point of view of any process $p \in A_i$, each execution of the same algorithm $\mathcal{A}$ in the system that consists in the processes of A_i and that is equipped with failure detectors Ω^x and Σ is indistinguishable from an execution $e \in \mathcal{E}_i$. Therefore, $\mathcal{A}$ can be used to solve the k'-set agreement problem in $\mathcal{MP}_{|A_i|,|A_i|-1}[\Omega^x, \Sigma]$, for some $k' < x$. However, as $2x \leq |A_i|$ this contradicts Lemma 12.

Let τ_i be a time at which every process in A_i has failed or have decided in execution e_i and define $\tau = \max\{\tau_i, 1 \leq i \leq n\}$. Let e be an execution of $\mathcal{A}$ that any process $p \in A_i$ cannot distinguish until time τ from the execution e_i, for each $i, 1 \leq i \leq z$. In particular, for every pair $\{p, q\}, p \in A_i, q \in A_j, i \neq j$, no message sent by p to q is delivered to q before time τ. Nevertheless, per the definition of the execution e_i, any collection of $z+1$ sets output by the failure detector Σ_z contain two intersecting sets. Therefore e is a legal execution of $\mathcal{A}$ in system $\mathcal{MP}_{|A_i|,|A_i|-1}[\Omega^x, \Sigma]$. The total number of distinct decided values in e is at least zx, hence $k \geq zx$. □

Given a failure detector Ω^x, it is easy to simulate a anti$-\Omega^x$ failure detector by outputting the complement of the sets leader output by Ω^x. Therefore,

Corollary 1. *Let $k, 1 \leq k \leq$ and $x, z, 1 \leq 2xz \leq n$. If $k < xz$, there is no k-set agreement algorithm in $\mathcal{MP}_{n,n-1}[\text{anti}-\Omega^x, \Sigma_z]$.*

Bonnet and Raynal introduce in [4] the failure detector class Π_k as a weakest failure detector candidate for message passing k-set-agreement. Next corollary disproves this conjuncture.

Corollary 2. *Let $k, n : 1 < k < n-1$ and $2k^2 \leq n$. There is no k-set agreement algorithm in $\mathcal{MP}_{n,n-1}[\Pi_k]$.*

Proof. [4] proves that Π_k is equivalent to $\Sigma_k \times \Omega^k$. The corollary then directly follows from Lemma 13 □

6 Concluding remarks

The paper has investigated the computational power of the failure detector classes Σ_x and anti$-\Omega^z$ as far as k-set-agreement is concerned in the n-processes message passing asynchronous model. The main result is that for large enough values of n, namely $n > 2kz$, k-set agreement is possible if and only if $k \geq xz$.

The main open question is the weakest failure detector for message passing k-set-agreement, for $1 < k < n-1$. Our xz-set agreement algorithm may help to demonstrate the sufficiency of weakest failure detector candidate. Another interesting avenue for future research is the complexity of k-set-agreement tolerating $t > n/2$ failures. When a majority of processes does not fail, it has been shown that the price of indulgence is constant [1, 15]. Is it still true when a majority of processes failures has to be tolerated?

References

[1] D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers. Of choices, failures and asynchrony: The many faces of set agreement. In *ISAAC 2009*, LNCS # 5878, pp. 943–953.

[2] A. F. Anta, S. Rajsbaum, and C. Travers. Brief announcement: weakest failure detectors via an egg-laying simulation. In *PODC 2009*, ACM Press, pp. 290–291.

[3] A. F. Anta, S. Rajsbaum, and C. Travers. Weakest failure detectors via an egg-laying simulation (preliminary version). Technical report, Universidad Rey Juan Carlos. Reports on Systems and Communications, vol IX, no 2, Jan 2009. `http://gsyc.es/tr-docs/RoSaC-2009-2.pdf`.

[4] F. Bonnet and M. Raynal. Looking for the weakest failure detector for k-set agreement in message-passing systems: Is π_k the end of the road? In *SSS 2009*, LNCS # 5873, pp. 149–164.

[5] F. Bonnet and M. Raynal. A simple proof of the necessity of the failure detector sigma to implement an atomic register in asynchronous message-passing systems. *Information Processing Letters*, 110(4):153–157, 2010.

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

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

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

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

[10] W. Chen, J. Zhang, Y. Chen, and X. Liu. Weakening failure detectors for k-set agreement via the partition approach. In *DISC 2007*, LNCS # 4731, pp. 123–138.

[11] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. Sharing is harder than agreeing. In *PODC 2008*, ACM press, pp. 85–94.

[12] 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 *PODC 2004*, ACM press, pp. 338–346.

[13] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and A. Tielmann. The weakest failure detector for message passing set-agreement. In *DISC 2008*, LNCS # 5218, pp. 109–120.

[14] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and A. Tielmann. The disagreement power of an adversary: extended abstract. In *PODC 2009*, ACM Press, pp. 288–289.

[15] P. Dutta and R. Guerraoui. The inherent price of indulgence. *Distributed Computing*, 18(1):85–98, 2005.

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

[17] E. Gafni and P. Kuznetsov. The weakest failure detector for solving k-set agreement. In *PODC 2009*, ACM Press, pp. 83–91.

[18] R. Guerraoui. Indulgent algorithms (preliminary version). In *PODC 2000*, ACM Press, pp. 289–297.

[19] R. Guerraoui, M. Herlihy, P. Kuznetsov, N. A. Lynch, and C. C. Newport. On the weakest failure detector ever. *Distributed Computing*, 21(5):353–366, 2009.

[20] R. Guerraoui and N. A. Lynch. A general characterization of indulgence. *Transactions on Autonomous and Adaptive Systems*, 3(4), 2008.

[21] R. Guerraoui and M. Raynal. The alpha of indulgent consensus. *The Computer Journal*, 50(1):53–67, 2007.

[22] V. Hadzilacos and S. Toueg. Reliable broadcast and related problems. *Distributed Systems*, pp. 97–145, 1993.

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

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

[25] A. Mostéfaoui, S. Rajsbaum, M. Raynal, and C. Travers. On the computability power and the robustness of set agreement-oriented failure detector classes. *Distributed Computing*, 21(3):201–222, 2008.

[26] A. Mostéfaoui and M. Raynal. k-set agreement with limited accuracy failure detectors. In *PODC 2000*, ACM Press, pp. 143–152.

[27] G. Neiger. Failure detectors and the wait-free hierarchy. In *PODC 1995*, ACM Press, pp. 100–109.

[28] M. Raynal and C. Travers. In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. In *OPODIS 2006*, LNCS # 4305, pp. 3–19.

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

[30] J. Yang, G. Neiger, and E. Gafni. Structured derivations of consensus algorithms for failure detectors. In *PODC 1998*, ACM Press, pp. 297–306.

[31] P. Zieliński. Anti-omega: the weakest failure detector for set agreement. In *PODC 2008*, ACM Press, pp. 55–64.