Simultaneous Consensus vs Set Agreement a Message-Passing Sensitive Hierarchy of Agreement Problems

Michel Raynal, Julien Stainer

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 8236 – February 2013

Simultaneous Consensus vs Set Agreement a Message-Passing Sensitive Hierarchy of Agreement Problems

Michel Raynal[*] [**] Julien Stainer[**]

Abstract: This paper investigates the relation linking the s-simultaneous consensus problem and the k-set agreement problem. To this end, it first defines the (s, k)-SSA problem which captures jointly both problems: each process proposes a value, executes s simultaneous instances of the k-set agreement problem, and has to decide a value so that no more than sk different values are decided. The paper introduces then a new failure detector class denoted $Z_{s,k}$, which is made up of two components, one focused on the "shared memory object" that allows the processes to cooperate, and the other focused on the liveness of (s, k)-SSA algorithms. A novelty of this failure detector lies in the fact that the definition of its two components are intimately related. Then, the paper presents a $Z_{s,k}$-based algorithm that solves the (s, k)-SSA problem, and shows that the "shared memory"-oriented part of $Z_{s,k}$ is necessary to solve the (s, k)-SSA problem (this generalizes and refines a previous result that showed that the failure detector Σ_k is necessary to solve k-set agreement). Finally, the paper investigates the structure of the family of (s, k)-SSA problems and introduces generalized (asymmetric) simultaneous set agreement problems in which the parameter k can differ in each underlying k-set agreement instance. Among other points, it shows that, for $s, k > 1$, (a) the $(sk, 1)$-SSA problem is strictly stronger that the (s, k)-SSA problem which is itself strictly stronger than the $(1, ks)$-SSA problem, and (b) there are pairs (s_1, k_1) and (s_2, k_2) such that $s_1k_1 = s_2k_2$ and (s_1, k_1)-SSA and (s_2, k_2)-SSA are incomparable.

Key-words: Asynchronous system, Distributed computing, Failure detector, Fault tolerance, Message-passing system, Quorum, Reduction, k-Set agreement, Simultaneous consensus, Wait-freedom

Consensus simultanés vs accord ensembliste : une hiérarchie de problèmes

Résumé : *Ce rapport etudie les liens entre les problèmes du consensus s-simultané et de l'accord k-ensembliste. Il commence par définir le problème (s, k)-SSA qui généralise les deux problèmes précédents. Le rapport introduit ensuite un nouveau détecteur de fautes, noté $Z_{s,k}$, formé de deux composantes, l'une chargé de rendre possible l'implémentation d'une forme d'"objet mémoire partagée" permettant aux processus de coopérer, tandis que la seconde fournit la vivacité nécessaire à un algorithme de (s, k)-SSA. Une particularité de ce détecteur de fautes réside dans le fait que les définitions de ces deux composantes sont intimement liées. Le rapport présente un algorithme basé sur $Z_{s,k}$ qui résoud le problème (s, k)-SSA et prouve que la partie orientée "mémoire partagée" de $Z_{s,k}$ est nécessaire pour le résoudre Enfin, le rapport étudie la structure de la famille des problèmes (s, k)-SSA et montre entre autres que pour $s, k > 1$, (a) le problème $(sk, 1)$-SSA est strictement plus difficile que le problème (s, k)-SSA, lui-même strictement plus difficile que le problème $(1, ks)$-SSA et (b) il existe des paires (s_1, k_1) et (s_2, k_2) avec $s_1k_1 = s_2k_2$ telles que les problèmes (s_1, k_1)-SSA et (s_2, k_2)-SSA sont incomparables.*

Mots clés : *système asynchrone, calcul réparti, détecteur de fautes, tolérance aux pannes, système à passage de messages, quorum, réduction, accord k-ensembliste, consensus simultanés, sans attente*

[*] Institut Universitaire de France
[**] ASAP : équipe commune avec l'Université de Rennes 1 et Inria

1 Introduction

The k-set agreement problems The k-set agreement problem is a paradigm of coordination problems. Defined in the setting of systems made up of processes prone to crash failures, it is a simple generalization of the consensus problem (that corresponds to the case $k = 1$). The aim of this problem, introduced by Chaudhuri [9], was to investigate how the number of choices (k) allowed to the processes is related to the maximum number of processes t that can crash. The problem is defined as follows. Each process proposes an input value, and any process that does not crash must decide a value (termination), such that a decided value is a proposed value (validity), and no more than k distinct values are decided (agreement).

While it can be solved in synchronous systems prone to any number of process crashes (see [19] for a survey), the main result associated with k-set agreement is the impossibility to solve it in presence of both asynchrony and process crashes when $t \geq k$ [4, 14, 23].

A way to circumvent this impossibility consists in enriching the underlying pure asynchronous system with a failure detector [7, 22]. A failure detector is a device that provides processes with information on failures. According to the type and the quality of this information, several failure detectors have been proposed (see [20] for a survey of failure detectors suited to k-set agreement). It has been shown that the failure detector $\overline{\Omega}_k$ (anti-omega-k) [18, 24] is the weakest failure detector that allow k-set agreement to be solved despite any number of process crashes in asynchronous *read/write* systems [12].

The situation is different in asynchronous crash-prone *message-passing* system. More precisely, (a) while weakest failure detectors are known only for the cases $k = 1$ and $k = n-1$ [8, 10, 11], (b) it has been shown that the generalized quorum failure detector denoted Σ_k is necessary [3]. k-Set agreement algorithms based on failure detectors stronger than Σ_k can found in [3, 5, 15, 16, 17].

The s-simultaneous consensus problem This problem has been introduced in [1]. Each of the n processes proposes the same value to s independent instances of the consensus problem, denoted $1, ..., s$. Each correct process has to decide a pair (c, v) (termination), where $c \in \{1, ..., s\}$ is a consensus instance and v is a proposed value (validity). Moreover, if (c, v) and (c, v') are decided we have $v = v'$ (agreement). (This is the scalar form of the problem: each process proposes the same value to each consensus instance. In the vector form, a process proposes a vector of s values, one value to each consensus instance. It is shown in [1] that both forms have the same computational power).

It is shown in [1] that the x-simultaneous consensus problem and the x-set agreement problem are computationally equivalent in asynchronous read/write systems where up to $t = n - 1$ processes may crash. It follows that in these systems, the failure detector $\overline{\Omega}_x$ is both necessary and sufficient to solve x-simultaneous consensus.

As far asynchronous message-passing systems are concerned, it is shown in [6] that, for $x > 1$ and $t > \frac{n+x-2}{2}$, x-simultaneous consensus is strictly stronger than x-set agreement. This means that, differently from what can be done in asynchronous read/write systems, it is not possible to solve x-simultaneous consensus from a black box solving x-set agreement.

Content of the paper The aim of this paper is to (a) better understand the relations linking s-simultaneous consensus and k-set agreement, and (b) near the weakest failure detector that allows k-set agreement to be solved in crash-prone asynchronous message-passing system.

To this end, the paper introduces first a problem that generalizes both s-simultaneous consensus and k-set agreement. This problem, denoted (s, k)-SSA (for s-Simultaneous k-Set Agreement) consists in s independent instances of the k-set agreement problem (hence, $(s, 1)$-SSA is x-simultaneous consensus, while $(1, k)$-SSA is k-set agreement).

Then, the paper introduces a new failure detector, denoted $Z_{s,k}$, that allows (s, k)-SSA to be solved in the asynchronous message-passing communication model, despite any number of process crashes. This failure detector is captured by an array of size s whose each entry is made up of two components. The first, which is nothing else than the *quorum* failure detector Σ_k, addresses the data sharing needed to correctly coordinate the processes. The second component states a leader-based property that allows the correct processes to always decide a value. When considering the $(1, k)$-SSA problem, it appears that $Z_{1,k}$ is a weaker failure detector than all the failure detectors proposed so far to solve k-set agreement. A noteworthy feature of $Z_{s,k}$ lies in the fact that these two components are not defined

independently one from the other (e.g., as done in the pair (Σ, Ω) [10]), namely, the definition of the *leader* component of *some* entry of the array is intimately related to the associated *quorum* component.

The paper presents then a $Z_{s,k}$-based algorithm that solves the (s, k)-SSA problem, and shows that the quorum part of $Z_{s,k}$ is necessary to solve the (s, k)-SSA problem (this proof generalizes the proof given in [3] that shows that Σ_k captures information on process crashes that is necessary to solve k-set agreement).

Last but not least, the paper considers the family of asymmetric $\{k_1, ..., k_s\}$-SSA problems, defined by s simultaneous instances of the k_x-set agreement problem where $k_x = k_1, ..., k_s$. It shows that these problems define a strong hierarchy from a computability point of view. It follows from this hierarchy that (as indicated in the abstract) for $s, k > 1$, (a) the $(sk, 1)$-SSA problem is strictly stronger that the (s, k)-SSA problem which is itself strictly stronger than the $(1, ks)$-SSA problem, and (b) there are pairs (s_1, k_1) and (s_2, k_2) such that $s_1k_1 = s_2k_2$ and (s_1, k_1)-SSA and (s_2, k_2)-SSA are incomparable problems. More generally, given K, the paper shows that the structure of the set of symmetric (s, k)-SSA problems (where $sk = K$) is a lattice where an arrow from A to B means that B can be solved from a block box solving A, but not vice-versa. The paper associates also with each such pair a failure detector that is necessary to solve A and a failure detector that is sufficient to solve B.

Roadmap The paper is made up of 6 sections. Section 2 defines the computation model, the (s, k)-SSA problem, and the Failure Detector class $Z_{s,k}$. Section 3 presents a simple $Z_{s,k}$-based algorithm that solves the (s, k)-SSA problem. Section 4 proves that the safety part of $Z_{s,k}$ is necessary when one wants to solve the (s, k)-SSA problem (from information on failures). Section 5 investigates the graph structure of the family of asymmetric SSA problems and shows that these problems define a strong hierarchy. Finally Section 6 concludes the paper.

2 Computation Model, (s, k)-SSA Problem, and the Failure Detector $Z_{s,k}$

2.1 Computation model

Process model The system is made up of n asynchronous sequential processes denoted $\Pi = \{p_1, \ldots, p_n\}$ (to simplify notations, we sometimes consider that Π is the set $\{1, \ldots, n\}$). Each process is a Turing machine enriched with two operations, which allows it to send and receive messages. "Asynchronous" means that there is no assumption on the speed of processes: each process proceeds at its own speed, which may arbitrarily vary and is unknown from the other processes.

A process behaves correctly until it possibly crashes (a crash is an unanticipated premature stop). Up to $t < n$ processes may crash. The case $t = n - 1$ defines the *wait-free* model. A process that crashes in a run is said to be *faulty* in that run, otherwise, it *correct*. Given a run, C denotes the set of processes which are correct in that run.

Communication model Each pair of processes is connected by a bidirectional channel. The channels are failure-free (no creation, duplication, alteration, or loss of messages), and asynchronous. "Asynchronous" means that, while each message is received, there is no bound on message transfer delays.

Timing model The underlying timing model is the set of natural integers $\mathbb{N}$. As the system is asynchronous, this time notion remains unknown to the processes. It is only used, from an external observer point of view, to state or prove properties. Time instants are denoted τ, τ', etc.

Notation The previous system is denoted $\mathcal{AMP}_{n,t}[\emptyset]$. Hence, $\mathcal{AMP}_{n,n-1}[\emptyset]$ denotes the wait-free message-passing model.

2.2 The s-Simultaneous k-Set Agreement –(s, k)-SSA– Problem

As indicated in the introduction, the s-simultaneous k-set agreement problem (in short (s, k)-SSA) consists in the simultaneous execution of s instances of the k-set agreement problem. Moreover, each process proposes the same value to each instance of the k-set agreement problem. The (s, k)-SSA problem is defined by the three following properties.

- Termination. Every correct process decides.
- Validity. A decided value is a pair (c, v) where $1 \leq c \leq s$ and v is a value proposed by a process.
- Agreement. For any $c \in \{1, ..., s\}$, there are at most k different values v such that (c, v) is decided.

It is easy to see that at most $K = sk$ different values v are decided, and consequently, any algorithm solving the (s, k)-SSA problem solves the K-set agreement problem. Moreover, $(1, k)$-SSA is k-set agreement, while $(s, 1)$-SSA is s-simultaneous consensus.

2.3 The Failure Detector Class $Z_{s,k}$

Definition A failure detector of the class $Z_{s,k}$ provides each process p_i with two arrays denoted $qr_i[1..s]$ and $\ell d_i[1..s]$. Intuitively, $qr_i[z]$ and $\ell d_i[z]$, $1 \leq z \leq s$, denote, with respect to the index z, the current quorum and the current leader of p_i, respectively. $Z_{s,k}$ is defined by the following properties, where $qr_i^\tau[z]$ and $\ell d_i^\tau[z]$ denote the value of $qr_i[z]$ and $\ell d_i[z]$ at time τ.

- Safety property. $\forall\ z \in [1..s]$:
  - Quorum intersection property (QI).
    $\forall\ i_1, ..., i_{k+1} \in \Pi,\ \forall\ \tau_1, ..., \tau_{k+1} : \exists h, \ell \in [1..k+1] :\ (h \neq \ell) \wedge (qr_{i_h}^{\tau_h}[z] \cap qr_{i_\ell}^{\tau_\ell}[z] \neq \emptyset)$.
  - Leader validity property(LV). $\forall\ \tau,\ \forall\ i : \ell d_i^\tau[z] \in \Pi$.
- Liveness property. $\exists z \in [1..s]$:
  - Quorum liveness property (QL). $\forall\ i \in C : \exists\ \tau : \forall\ \tau' \geq \tau : qr_i^{\tau'}[z] \subseteq C$.
  - Eventual leadership property (EL). $\exists \ell \in C :\ \forall i \in C :$
    $\big[\forall\ \tau : \exists\ \tau', \tau'' \geq \tau : (qr_i^{\tau'}[z] \cap qr_\ell^{\tau''}[z] \neq \emptyset)\big] \ \Rightarrow\ \big[\exists\ \tau : \forall\ \tau' \geq \tau : (\ell d_i^{\tau'}[z] = \ell)\big]$.

The quorum intersection property states that, for any $z \in \{1, ..., s\}$, there are two quorum values that intersect in any set of $k + 1$ quorum values, each taken at any time. The leader validity property states that the leader domain is the set of processes.

While the safety properties concern all the entries of the arrays $qr_i[1..s]$ and $\ell d_i[1..s]$, the liveness properties are only on a single of these entries, say z. The quorum liveness property states that there is a finite time after which all quorum values (appearing in $qr_i[z]$ for every $i \in C$) contain only correct processes. The eventual leader liveness property involves only the quorum values taken by the entries $qr_i[z]$, for every $i \in C$. Hence, it relates these quorum values with the eventual leader values in the local variables $\ell d_i[z]$ at each correct process p_i. More precisely, it states that there is a correct process p_ℓ such that, for any correct process p_i whose quorum $qr_i[z]$ intersects infinitely often with the quorum $qr_\ell[z]$ of p_ℓ (left part of the implication), p_ℓ becomes eventually the permanent leader of p_i (saved in $\ell d_i[z]$, right part of the implication).

Notation Let $Z(Q)_{s,k}$ denote the quorum part of $Z_{s,k}$ (defined by the properties QI and QL). Similarly, let $Z(L)_{s,k}$ denote the leader part of $Z_{s,k}$ (defined by the properties LV and EL where the quorum part brings no information on failures, which means that we have then $\forall\ i, \forall\ z,\ \forall \tau : qr_i^\tau[z] = \Pi$).

Particular cases This part shows the generality of $Z_{s,k}$ by displaying failure detector classes that have appeared in the literature. As they are not important for the paper, their initial definitions are not recalled here (the reader will find them in the associated referenced papers).

- $s = k = 1$: $Z_{1,1} = (\Sigma, \Omega)$ as defined in [8, 10].
- $Z_{s,k}$ is weaker than the failure detector $\Pi\Sigma_{k,s}$ introduced in [17].
- $Z_{s,1}$ is the failure detector W_s as defined in [6].
- $Z(Q)_{1,k}$ is the quorum failure detector Σ_k introduced in [3].
- $Z(Q)_{s,1}$ is the failure detector $V\Sigma_s$ introduced in [6] where it is shown that (a) $V\Sigma_s$ (i.e., $Z(Q)_{s,1}$) is necessary to solve the s-simultaneous consensus problem, and (b) $(V\Sigma_s, \overline{\Omega}_s) \preceq W_s (= Z_{s,1}) \preceq (V\Sigma_s, \Omega)$.
- $Z(Q)_{1,n-1}$ is the quorum failure detector $\mathcal{L}$ introduced in [11]. This follows from the fact that $Z(Q)_{1,n-1}$ is Σ_{n-1} and the equivalence between Σ_{n-1} and $\mathcal{L}$ established in [16].
- $Z(L)_{s,1}$ is the failure detector $\overline{\Omega}_k$ introduced in [18, 24] and used in [12].

Notation $\mathcal{AMP}_{n,n-1}[FD]$ denotes the wait-free message-passing model enriched with a failure detector of the class FD.

3 A $Z_{s,k}$-based Algorithm for the (s, k)-SSA problem

This section presents a simple algorithm that solves the (s, k)-SSA problem in $\mathcal{AMP}_{n,n-1}[Z_{s,k}]$. This algorithm consists in the concurrent execution of s algorithms, each solving an instance of the k-set agreement problem (i.e., an instance of the $(1, k)$-SSA problem). This algorithm is based on an underlying abstraction (object) called alpha_k.

3.1 The Abstraction alpha_k

Historical perspective The abstraction alpha_k originates from a similar abstraction (called alpha) introduced in [13] (see also [21]) to capture the safety property of the consensus problem in message-passing systems. It has then been generalized to capture the safety property of k-set agreement in (a) read/write systems in [22], and (b) message-passing systems in [6, 17].

Definition We consider here the alpha_k object that we have introduced in [17]. Let $\perp$ be a default value that cannot be proposed by processes. alpha_k is an object, initialized to $\perp$, that may store up to k different values proposed by processes. It is an abstraction (object) that provides processes with a single operation denoted $\mathsf{alpha\_propose}_k(r, v)$ (where r is a round number and v a proposed value), which returns a value to the invoking process. The round number plays the role of a logical time that allows identifying the $\mathsf{alpha\_propose}_k()$ invocations. It is assumed that distinct processes use different round numbers and successive invocations by the same process use increasing sequence numbers. alpha_k is an *abortable* object in the sense that $\mathsf{alpha\_propose}_k()$ invocations are allowed to return the default value $\perp$ (i.e., abort) in specific concurrency-related circumstances (as defined from the obligation property, see below). More precisely, the alpha_k objects used in this paper are defined by the following specification in which the obligation property takes explicitly into account the fact that these objects are used in the system model $\mathcal{AMP}[Z_{1,k}]$ (which is strictly stronger than $\mathcal{AMP}[\emptyset]$). The properties defining such an alpha_k object are the following.

- Termination. Any invocation of $\mathsf{alpha\_propose}_k()$ by a correct process terminates.
- Validity. If $\mathsf{alpha\_propose}_k(r, v)$ returns $v' \neq \perp$, then $\mathsf{alpha\_propose}_k(r', v')$ has been invoked with $r' \leq r$.
- Quasi-agreement. At most k values different from the default value $\perp$ can be returned by the invocations of $\mathsf{alpha\_propose}_k()$.
- Obligation. (As $s = 1$, $qr_i[1]$ is denoted qr_i.) p_ℓ being a correct process, let $Q(\ell, \tau) = \{i \in C \mid \forall\, \tau_i, \tau_\ell \geq \tau : qr_i^{\tau_i} \cap qr_\ell^{\tau_\ell} = \emptyset\}$. If, after some finite time τ, (a) only p_ℓ and processes in $Q(\ell, \tau)$ invoke $\mathsf{alpha\_propose}_k()$ and (b) p_ℓ invokes $\mathsf{alpha\_propose}_k()$ infinitely often, then at least one invocation issued by p_ℓ returns a non-$\perp$ value.

Differently from the obligation property stated in [5, 13, 22] the previous obligation property is $Z_{s,k}$-aware (or more precisely $Z_{1,k}$-aware, i.e., Σ_k-aware). This obligation property allows concurrent invocations of $\mathsf{alpha\_propose}_k()$ to return non-$\perp$ values as soon as the quorums of the invoking processes do not intersect during these invocations. An algorithm implementing the previous alpha_k object in $\mathcal{AMP}[\Sigma_k]$ is described in [17].

3.2 A Base Algorithm for the $(1, k)$-SSA Problem (k-Set Agreement)

Algorithm 1 is a very simple algorithm solving the k-set agreement problem in $\mathcal{AMP}[Z_{1,k}]$. A process p_i invokes $\mathsf{ssa\_propose}_{1,k}\ (v_i)$ where v_i is the value it proposes. It decides a value v when it executes the statement $\mathsf{return}\ (v)$ which terminates its invocation. The local variable r_i is the local round number (as it is easy to see, each process uses increasing round numbers and no two different processes use the same round numbers).

A process loops until it decides. If during a loop iteration p_i is such that $\ell d_i = i$ (where ℓd_i denotes $\ell d_i[1]$, the single leader entry locally output at p_i by $Z_{1,k}$), p_i invokes the underlying alpha_k distributed object (denoted

```
operation ssa_propose_{1,k} (v_i):
(01) dec_i ← ⊥; r_i ← i;
(02) while (dec_i = ⊥) do
(03)    if (ℓd_i = i) then dec_i ← ALPHA_k.alpha_propose_k(r_i, v_i); r_i ← r_i + n end if
(04) end while;
(05) for each j ∈ {1, ..., n} do send DECISION(dec_i) to p_j end for; return (v).

when DECISION(v) is received:
(06) for each j ∈ {1, ..., n} do send DECISION(dec_i) to p_j end for; return (v).
```

Algorithm 1: k-Set agreement in $\mathcal{AMP}[Z_{1,k}]$ (code for p_i)

$ALPHA_k$) to try to deposit its value v_i into it (the success depends on the concurrency and quorums pattern). If a non-$\perp$ value is returned by this invocation, p_i broadcasts it and decides (execution of return()). If it has not yet decided, a process decides when it receives a DECISION() message (lines 5-6 implement a reliable broadcast).

Theorem 1 *Algorithm 1 solves the k-set agreement problem in $\mathcal{AMP}[Z_{1,k}]$.*

Proof Validity and agreement properties of k-set agreement. Let us first observe that, due to the test of line 2, the default value $\perp$ cannot be decided. The fact that a decided value is a proposed value follows then from the validity of the underlying alpha$_k$ object. Similarly, the fact that at most k non-$\perp$ values are decided follows directly from the quasi-agreement property of the underlying alpha$_k$ object.

Termination property of k-set agreement. It follows from lines 5 and 6 that, at soon as a process decides (invokes return()) each correct process eventually delivers the same DECISION(v) message and decides (if not yet done). The proof is by contradiction: assuming that no process decides, we show that at least one correct process executes line 5 (and consequently, all correct processes decide).

Let p_ℓ be the correct process that appears in the definition of the eventual leadership property of $Z_{1,k}$. It follows from the definition of p_ℓ that we eventually have forever $\ell d_\ell = \ell$.

Let R_ℓ be the (possibly empty) set of identities of the processes p_j (with $j \neq \ell$) such that we have $\ell d_j = j$ infinitely often. It follows from the contrapositive of the eventual leadership property of $Z_{1,k}$ that there is a time τ_{R_ℓ} such that $\forall\, j \in R_\ell, \forall\, \tau 1, \tau 2 \geq \tau_{R_\ell}$: $qr_j^{\tau 1} \cap qr_\ell^{\tau 2} = \emptyset$, from which we conclude that $R_\ell \subseteq Q(\ell, \tau_{R_\ell})$ (this is the set defined in the obligation property defining an alpha$_k$ object).

Let us notice that, due to test of line 3, there is a finite time τ_a after which the only processes that invoke alpha_propose$_k$() are the processes in $R_\ell \cup \{\ell\}$. Moreover (as by the contradiction assumption no process decides) it follows that, after τ_a, p_ℓ invokes alpha_propose$_k$() infinitely often. Let τ_b be a time greater than $\max(\tau_{R_\ell}, \tau_a)$ from which we have $R_\ell \subseteq Q(\ell, \tau_{R_\ell}) \subseteq Q(\ell, \tau_b)$.

As after τ_b (a) only processes in $R_\ell \cup \{\ell\}$ invoke alpha_propose$_k$(), (b) p_ℓ invokes alpha_propose$_k$(), infinitely often, and (c) $R_\ell \subseteq Q(\ell, \tau_b)$, we conclude from the obligation property of the alpha$_k$ object that at least one invocation of p_ℓ returns a value $v \neq \perp$ and consequently sends the message DECISION(v) to all the processes. This contradicts the fact that no process decides and concludes the proof of the theorem. $\square_{Theorem\ 1}$

3.3 An Algorithm for the (s, k)-SSA Problem

A simple algorithm solving the (s, k)-SSA problem can be easily obtained by launching s concurrent instances of algorithm 1, the zth instance ($1 \leq z \leq s$) relying, at each process p_i, on the components $qr_i[z]$ and $\ell d_i[z]$ of $\mathcal{AMP}[Z_{s,k}]$. A process decides the value returned by the first of the s instances that locally terminates. Hence, it decides the pair (c, v) where c is its first deciding instance and v the value it decides in that instance. As there are s instances of algorithm 1 and at most k values can be decided in each of them, it follows that at most $K = sk$ different values can be decided. Moreover, as there is at least one instance z such that the failure detector outputs $\ell d_i[z]$ at each correct process p_i converge to the same correct process, it follows that the correct processes decide (if not done before) in at least one of the s instances of algorithm 1.

4 $Z(Q)_{s,k}$ is Necessary to Solve the (s, k)-SSA problem

This section shows that $Z(Q)_{s,k}$ is necessary to solve the (s, k)-SSA problem as soon as we are looking for a failure detector-based solution. To that end, given an algorithm A that solves the (s, k)-SSA problem in $\mathcal{AMP}_{n,n-1}[FD]$, this section presents an algorithm that emulates the output of $Z(Q)_{s,k}$, namely an array $qr_i[1..s]$ at each process p_i, which satisfies the properties QI and QL. This means that it is possible to build $Z(Q)_{s,k}$ from any failure detector FD that can solve the (s, k)-SSA problem.

According to the usual terminology, $Z(Q)_{s,k}$ is *extracted* from the FD-based algorithm A. This extraction is a generalization of the algorithm introduced in [3], which extracts Σ_k from any failure detector-based algorithm that solves the k-set agreement problem.

The Extraction Algorithm Each process p_i participates in several executions of the algorithm A. S being a set of processes, A^S denotes the execution of A in which exactly the processes of S participate. In this execution, each process of S either decides, blocks forever, or crashes. So the execution of the extraction algorithm is composed of $2^n - 1$ executions of A.

Init: $Q_i[1, \ldots, s] \leftarrow [\Pi, \ldots, \Pi]$; $queue_i \leftarrow \langle 1, \ldots, n\rangle$;
for each $S \subseteq \Pi$ **such that**$(i \in S)$ **do** A^S.ssa_propose$_{s,k}(i)$ **end for**; activate the tasks $T1$ to $T5$.

Task $T1$: **repeat periodically** send ALIVE(i) to each p_j such that $j \in \Pi \setminus \{i\}$ **end repeat**.

Task $T2$: **when** ALIVE(j) **is received**: move j at the head of $queue_i$.

Task $T3$: **when** $(c, -)$ **is decided by** p_i **in the** cth k-set agreement instance of A^S:
$Q_i[c] \leftarrow Q_i[c] \cup \{S\}$; send DECISION$(c, S)$ to each p_j such that $j \in \Pi \setminus \{i\}$.

Task $T4$: **when** DECISION(c, S) **is received**: $Q_i[c] \leftarrow Q_i[c] \cup \{S\}$.

Task $T5$: **repeat forever**
for each $c \in \{1, ..., s\}$ **do**
$min\_rank_i \leftarrow \min\{\max\{rank(queue_i, j), j \in S\}, S \in Q_i[c]\}$;
$qr_i[c] \leftarrow$ any $S_{min} \in Q_i[c]$ such that $\max\{rank(queue_i, j), j \in S_{min}\} = min\_rank_i$
end for;
end repeat.

Algorithm 2: Extracting $Z(Q)_{s,k}$ from a failure detector-based algorithm A solving the (s, k)-SSA problem

The behavior of each process p_i is described in algorithm 2. The internal statements of the tasks $T1$ and $T5$, and the tasks $T2$-$T4$ are locally executed in mutual exclusion. The local array $Q_i[1..s]$ is initialized to $[\Pi, \ldots, \Pi]$. The aim of $Q_i[c]$ is to contain all the sets S such that a value has been decided in the cth instance of the k-set agreement of the execution of A^S.

Initially, each process p_i proposes its identity i to all the instances of A in which it participates. To that end it invokes A^S.ssa_propose$_{s,k}(i)$ for each set S such that $i \in S$ (ssa_propose$_{s,k}()$ is the operation associated with each instance of the (s, k)-SSA problem). When it decides in the cth k-set agreement of A^S (task $T3$), p_i adds the set S to $Q_i[c]$ and informs each other process p_j, which includes S in $Q_j[c]$ when it learns it (task $T4$).

Each alive process p_i sends periodically messages ALIVE(i) (task $T1$) to inform the other processes that it is alive. When it receives a message ALIVE(j) (task $T2$), a process p_i moves j at the head of its local queue (denoted $queue_i$) which always contains all process identities. It follows that the identities of all the correct processes eventually precede in this queue the identities of all the faulty processes. (Initially, each queue $queue_i$ contains all process identities, in any order.)

$T5$ is a task whose aim is to repeatedly compute the current value of $qr_i[1..s]$. It uses the function $rank(queue_i, j)$ which returns the current rank of p_j in the queue $queue_i$. The value of $qr_i[c]$ is computed as follows. It is the "first set of $Q_i[c]$ with respect to $queue_i$" (i.e., with respect to the processes which are currently seen as being alive). This is captured with the help of the local variable $minrank_i$. As an example Let $Q_i[c] = \{\{3, 4, 9\}, \{2, 3, 8\}, \{4, 7\}, \{1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$, and $queue_i = \langle 4, 8, 3, 2, 7, 5, 9, 1, 6\rangle$. We have then $minrank = 4$, and $S_{min} = \{2, 3, 8\}$. This set of identities is

the first set of $Q_i[c]$ with respect to $queue_i$ because each of the other sets $\{3,4,9\}$, $\{4,7\}$, or $\{1,2,3,4,5,6,7,8,9\}$, includes an element (9, 7, and 6, respectively) that appears in $queue_i$ after all the elements of $\{2,3,8\}$ (in case several sets are "first", any of them can be selected).

Theorem 2 *Given any algorithm A that solves the (s,k)-SSA problem in $\mathcal{AMP}_{n,n-1}[FD]$, the extraction algorithm described in Figure 2 is a wait-free construction of a failure detector $Z(Q)_{s,k}$.*

Proof Proof of the quorum liveness property. We have to show that there is an entry z such that, after some finite time and for each $i \in C$, $qr_i[z]$ contains only correct processes. Let us consider the execution of A^C. As all the processes in C are correct (definition), A^C terminates. It then follows from the tasks $T3$ and $T4$ that there is an entry z such the set $Q_i[z]$ of each correct process eventually contains the set C.

Moreover, as each correct process sends forever messages ALIVE() (task $T1$), it follows that the identities of the correct processes appear in queue $queue_i$ before the identities of the faulty processes (task $T2$). It then follows from task $T5$ (which computes the "first set of $Q_i[z]$ with respect to $queue_i$") that, after some finite time, $qr_i[z]$ contains only correct processes.

Proof of the quorum safety property. We have to show that, for each entry $z \in \{1,...,s\}$, we have $\forall\, i_1,...,i_{k+1} \in \Pi,\ \forall\, \tau_1,...,\tau_{k+1} : \exists h,\ell \in [1..k+1] :\ (h \neq \ell) \wedge (qr_{i_h}^{\tau_h}[z] \cap qr_{i_\ell}^{\tau_\ell}[z] \neq \emptyset)$. The proof is by contradiction.

Let us assume that it exists a family $(i_m)_{1\leq m\leq k+1}$ of $k+1$ (not necessarily distinct) process identities, a family $(\tau_m)_{1\leq m\leq k+1}$ of time instants and an integer $z \in \{1,\ldots,s\}$ such that for all $m_1, m_2 \in \{1,\ldots,k+1\}$ $qr_{i_{m_1}}^{\tau_{m_1}}[z] \cap qr_{i_{m_2}}^{\tau_{m_2}}[z] = \emptyset$. Let S_m denote the set $qr_{i_m}^{\tau_m}[z]$. Let us remark that the previous non-intersection assumption and the fact that no set S_m is empty imply that, for each $m \in \{1,\ldots,k+1\}$, a process $p_{j_m} \in S_m$ decided a value v_m in the zth k-set agreement of $\mathcal{A}^{S_m}$ (no set S_m can have the initial value Π since it would intersect any other one). As the sets $S_m, 1 \leq m \leq k+1$ are pairwise disjoint, there is an execution of $\mathcal{A}$ in which the set of participants is $\bigcup_{m\in\{1,\ldots,k+1\}} S_m$ that is indistinguishable from the considered execution of $\mathcal{A}^{S_m}$ from the point of view of each p_{j_m} (just consider that the messages between processes of S_{m_1} and S_{m_2} are delayed and received only after all processes decide). It follows that in this execution each process p_{j_m} also decides v_m in the zth instance of the k-set agreement. It then follows from the agreement property of the k-set agreement problem that $|\{v_m, 1 \leq m \leq k+1\}| \leq k$. Hence, it exists $m_1, m_2 \in \{1,\ldots,k+1\}$ such that $v_{m_1} = v_{m_2}$. But, since the values decided in the zth instance of the k-set agreement problems involved in both $\mathcal{A}^{S_{m_1}}$ and $\mathcal{A}^{S_{m_2}}$ are identities of participating processes (this follows from the validity property of the k-set agreement problem), it follows that $v_{m_1} = v_{m_2} \in S_{m_1} \cap S_{m_2}$. This contradicts the initial assumption, which concludes the proof of the quorum intersection property of $Z(Q)_{s,k}$. $\square_{Theorem\ 2}$

5 The Structure of Generalized (s,k)-SSA problems

This section studies the mathematical structure of the family of (s,k)-SSA problems for $sk = K$. To that end, it first introduces a straightforward generalization of this family and then shows that this generalized family can be represented by a directed graph where an arrow from A to B means that the problem B can be solved from a black box solving the problem A, while the opposite is impossible. Given such a pair of problems (A,B), this section also associates with this pair a failure detector that is necessary to solve A and a failure detector that is sufficient to solve B.

5.1 The Generalized Asymmetric $\{k_1,...,k_s\}$-SSA Problem

While the (s,k)-SSA problem is a symmetric problem which consists in s simultaneous instances of the k-set agreement problem, a simple generalization consists in considering an asymmetric version made up of s simultaneous instances of possibly different set agreement problems, namely the k_1-set agreement problem, the k_2-set agreement problem, etc., and the k_s-set agreement problem. Hence, among the proposed values, at most $K = \Sigma_{x=1}^{s} k_x$ different values are decided.

This asymmetric version is denoted $\{k_1, ..., k_s\}$-SSA where $\{k_1, ..., k_s\}$ is a multiset[1]. The particular instance where $k_1 = \cdots = k_s = k$ is the symmetric (s, k)-SSA problem. As permuting the integers k_x does not change the problem, we consider the canonical notation where $k_1 \geq k_2 \geq \ldots \geq k_s \geq 1$.

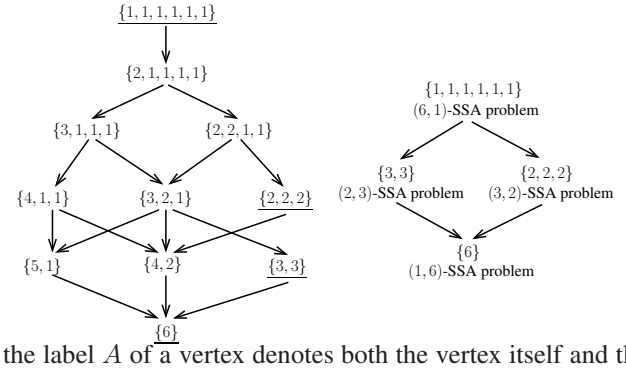
5.2 Associating a Graph with a Family of Generalized $\{k_1, ..., k_s\}$-SSA Problems

Graph definition Given an integer K and starting from the source vertex labeled with the multiset $\{1, ..., 1\}$ (K times the integer 1), let us define a graph denoted $G(K)$ as follows. Given a vertex labeled $\{k_1, ..., k_s\}$ (initially, $s = K$ and $k_1 = \cdots = k_K = 1$), we add all possible vertices of $s - 1$ elements labeled $\{k'_1, ..., k'_{s-1}\}$ and directed edges from $\{k_1, ..., k_s\}$ to each vertex $\{k'_1, ..., k'_{s-1}\}$ defined as follows. Any pair of elements k_x, k_y of the multiset $\{k_1, ..., k_s\}$ gives rise to a vertex labeled by the multiset $\{k'_1, ..., k'_{s-1}\}$ such that

$$\{k'_1, ..., k'_{s-1}\} \ = \ \{k_1, ..., k_s\} \setminus \{k_x, k_y\} \cup \{k_x + k_y\}.$$

Then, the construction process is recursively repeated until we arrive at a sink node composed of a single element labeled $\{K\}$.

An example of graph for $K = 6$ is given on the right. The labels corresponding to symmetric instances ((s, k)-SSA problems) are underlined. The graph (lattice) on the right side of the figure considers only the symmetric problem instances.



Meaning of the graph As we will see in Section 5.4, given an integer K, this graph describes the computability hierarchy linking all the $\{k_1, k_2, ...\}$-SSA agreement problems such that $k_1 + k_2 + \cdots = K$. Let the label A of a vertex denotes both the vertex itself and the associated agreement problem. An edge from a vertex A to a vertex B means that (a) given an algorithm that solves the problem A it is possible to solve the problem B, while (b) the opposite is impossible.

Lemma 1 *$G(K)$ is cycle-free.*

Proof The proof is an immediate consequence of the fact that the successors of any vertex labeled by a multiset of size s are multisets of size $s - 1$. $\square_{Theorem\ 1}$

As we will see in Lemma 2, the following predicate P characterizes (with the existence of a function f) the pairs of vertices connected by a path in $G(K)$.

Definition Let $\{k_1, ..., k_s\}$ and $\{k'_1, ..., k'_{s'}\}$ be any pair of vertices of $G(K)$.

$$P\big(\{k_1, ..., k_s\}, \{k'_1, ..., k'_{s'}\}\big) \stackrel{def}{=}$$
$$\exists f : \{1, \ldots, s\} \to \{1, \ldots, s'\} \text{ s.t. } \forall y \in \{1, \ldots, s'\} \ : k_y \ = \ \textstyle\sum_{x \in f^{-1}(y)} k_x.$$

Lemma 2 $\big(\exists$ *a path from* $\{k_1, ..., k_s\}$ *to* $\{k'_1, ..., k'_{s'}\}\big) \Leftrightarrow \big(P(\{k_1, ..., k_s\}, \{k'_1, ..., k'_{s'}\})\big)$.

Proof Direction $\Rightarrow$.
Let $\{a^1_1, \ldots, a^1_s\} \ = \ \{k_1, ..., k_s\}, ..., \{a^z_1, \ldots, a^z_{s-z+1}\}, ..., \{a^{s-s'+1}_1, \ldots, a^{s-s'+1}_{s'}\} \ = \ \{k'_1, ..., k'_{s'}\}$ be a path from $\{k_1, ..., k_s\}$ to $\{k'_1, ..., k'_{s'}\}$ in $G(K)$. (Hence, all the edges $(\{a^z_1, \ldots, a^z_{s-z+1}\}, \{a^{z+1}_1, \ldots, a^{z+1}_{s-z}\})$, $z \in \{1 \leq z \leq s - s'$, belong to $G(K)$.)

[1]The set notation is used to represent a multiset. A multiset is a set in which several elements can have the same value. As an example, $\{1, 2, 1, 1, 3\}$ is a multiset of 5 elements. Hence, the multisets $\{1, 2, 1, 1, 3\}$ and $\{2, 1, 3\}$ are different (while $\{1, 2, 1, 1, 3\} = \{2, 1, 3\}$ from a set point of view).

For each $z \in \{1, \ldots, s-s'\}$, it follows from the definition of $G(K)$ that there is a pair (x_z, y_z) of distinct integers in $\{1, \ldots, s-z+1\}$ such that $a^z_{x_z} \geq a^z_{y_z}$, $x_z < y_z$ (w.l.o.g.), and $\{a_1^{z+1}, \ldots, a_{s-z}^{z+1}\} = \{a_1^z, \ldots, a_{s-z+1}^z\} \setminus \{a^z_{x_z}, a^z_{y_z}\} \cup \{a^z_{x_z} + a^z_{y_z}\}$. Let f_z be the function that associates with each $x \in \{1, \ldots, s-z+1\}$ (i) x if $a^z_x > a^z_{x_z} + a^z_{y_z}$, (ii) $x+1$ if $a^z_{x_z} + a^z_{y_z} \geq a^z_x$ and $x < x_z$, (iii) x if $a^z_{x_z} + a^z_{y_z} \geq a^z_x$ and $x_z < x < y_z$, (iv) $x-1$ if $a^z_{x_z} + a^z_{y_z} \geq a^z_x$ and $y_z < x$, and (v) the index w_0 of the first occurrence of $a^z_{x_z} + a^z_{y_z}$ in $\{a_1^{z+1}, \ldots, a_{s-z}^{z+1}\}$ if $x \in \{x_z, y_z\}$. (The aim of these cases is to place the element $a^z_{x_z} + a^z_{y_z}$ at its right place in the multiset $\{a_1^{z+1}, \ldots, a_{s-z}^{z+1}\}$.)

By construction, for all $y \in \{1, \ldots, s-z\} \setminus \{w_0\}$, $f^{-1}(y)$ contains a single value and $a^z_{f^{-1}(y)} = a^{z+1}_y$, while $f^{-1}(w_0) = \{x_z, y_z\}$. Since $a^z_{x_z} + a^z_{y_z} = a^{z+1}_{w_0}$, $P(\{a_1^z, \ldots, a_{s-z+1}^z\}, \{a_1^{z+1}, \ldots, a_{s-z}^{z+1}\})$ is satisfied. Let f be $f_z \circ f_{z-1} \circ \cdots \circ f_1$, since the property verified by f is stable by composition, the path from $(\{k_1, ..., k_s\}$ to $\{k'_1, ..., k'_{s'}\})$ ensures that $P(\{k_1, \ldots, k_s\}, \{k'_1, ..., k'_{s'}\}$ is satisfied.

Direction $\Leftarrow$.
Let $(\{k_1, ..., k_s\}, \{k'_1, ..., k'_{s'}\})$ be a pair of vertices of $G(K)$ that satisfies the predicate P. Two cases are possible: (i) $s = s'$ and then f is the identity function (or just a reordering of the values appearing multiple times) and $\{k_1, ..., k_s\} = \{k'_1, \ldots k'_{s'}\}$ or (ii) $s > s'$.

In the case (i), there is a path (of length zero) from any node to itself. In the case (ii), consider the function f defined in the predicate P. Since $s > s'$, f is not injective (one-to-one), and there are two distinct integers x_1, y_1 such that $f(x_1) = f(y_1)$. Let then $\{a_1^2, \ldots, a_{s-1}^2\}$ be $\{k_1, ..., k_s\} \setminus \{k_{x_1}, k_{y_1}\} \cup \{k_{x_1+y_1}\}$. If f is injective from $\{1, \ldots, s\} \setminus \{y_1\}$ to $\{1, \ldots, s'\}$, since according to P it is surjective (onto), it follows that $s' = s-1$ and that $\{a_1^2, \ldots, a_{s-1}^2\} = \{k'_1, ..., k'_{s'}\}$ and there is a path in $G(K)$ from $\{k_1, ..., k_s\}$ to $\{k'_1, ..., k'_{s'}\}$. If f is not injective from $\{1, \ldots, s\} \setminus \{y_1\}$ to $\{1, \ldots, s'\}$, then it exists two distinct integers x_2, y_2 in $\{1, \ldots, s\} \setminus \{y_1\}$ such that $f(x_2) = f(y_2)$ (consider that $y_2 \neq x_1$, w.l.o.g.). Let then build $\{a_1^3, \ldots, a_{s-2}^3\}$ be (i) $\{k_1, ..., k_s\} \setminus \{k_{x_1}, k_{y_1}, k_{x_2}, k_{y_2}\} \cup \{k_{x_1} + k_{y_1}, k_{x_2} + k_{y_2}\}$ if $x_1 \neq x_2$, or (ii) $\{k_1, ..., k_s\} \setminus \{k_{x_1}, k_{y_1}, k_{y_2}\} \cup \{k_{x_1} + k_{y_1} + k_{y_2}\}$ if $x_2 = x_1$. If f is injective from $\{1, \ldots, s\} \setminus \{y_1, y_2\}$ to $\{1, \ldots, s'\}$ then, as it is also surjective, $s' = s-2$, $\{a_1^3, \ldots, a_{s-2}^3\} = \{k'_1, ..., k'_{s'}\}$ and there is a path of length 2 in $G(K)$ from $\{k_1, ..., k_s\}$ to $\{k'_1, ..., k'_{s'}\}$. If f is not injective from $\{1, \ldots, s\} \setminus \{y_1, y_2\}$ to $\{1, \ldots, s'\}$ then let choose x_3, y_3 in $\{1, \ldots, s\} \setminus \{y_1, y_2\}$ such that $f(x_3) = f(y_3)$ and $y_3 \notin \{x_1, x_2\}$ and continue the construction. After $s - s'$ iteration steps, we have $|\{1, \ldots, s\} \setminus \{y_1, \ldots, y_{s-s'}\}| = s'$, and the vertices which have been traversed belong to a path of $G(K)$ starting at $\{k_1, ..., k_s\}$ and ending at $\{k'_1, ..., k'_{s'}\}$. $\square_{Lemma\ 2}$

Theorem 3 *The transitive closure of $G(K)$ is a partial order.*

Proof The theorem follows from the fact that the relation captured by the predicate P is anti-symmetric (Lemma 1) and transitive (Lemma 2). $\square_{Theorem\ 3}$

5.3 Associated Generalized Failure Detector $GZ_{k_1,...,k_s}$

The failure detector $Z_{s,k}$ is implicitly tailored for the symmetric (s,k)-SSA problem. A simple generalization allows to extend it to obtain an "equivalent" failure detector suited to asymmetric problems.

As $Z_{s,k}$, this generalized failure detector, denoted $GZ_{k_1,...,k_s}$, provides each process p_i with an array $qr_i[1..s]$ and an array $\ell d_i[1..s]$. It differs from $Z_{s,k}$ in the constraint imposed by the quorum intersection property that is now specific to each entry $z \in \{1, ..., s\}$. More explicitly, QI is replaced by the property GQI defined as follows

- Quorum intersection property (GQI). $\forall\ z \in [1..s]$:
  $\forall\ i_1, ..., i_{k_z+1} \in \Pi,\ \forall\ \tau_1, ..., \tau_{k_z+1} : \exists h, \ell \in [1..k_z+1] :\ (h \neq \ell) \wedge (qr^{\tau_h}_{i_h}[z] \cap qr^{\tau_\ell}_{i_\ell}[z] \neq \emptyset)$.

The other properties –leader validity (LV), quorum liveness (QL), and eventual leader liveness (EL)– remain unchanged. It is easy to see, that $GZ_{k_1,...,k_s}$ boils down to $Z_{s,k}$ when $k_1 = \cdots = k_s = k$.

Let $GZ(Q)_{k_1,...,k_s}$ denotes the quorum part of $GZ_{k_1,...,k_s}$ (properties GQI and QL). The proof of the following theorem is a simple extension of the proof of Theorem 2. It is left to the reader.

Theorem 4 *Given any algorithm A that solves the $\{k_1, ..., k_s\}$-SSA problem in $\mathcal{AMP}_{n,n-1}[FD]$, the extraction algorithm described in Figure 2 is a wait-free construction of a failure detector $GZ(Q)_{k_1,...,k_s}$.*

5.4 A Hierarchy of Agreement Problems

Problem hierarchy Let $\mathcal{AMP}_{n,n-1}[X]$ denote the asynchronous message-passing model in which any number of processes may crash ($\mathcal{AMP}_{n,n-1}[\emptyset]$) enriched with an algorithm that solves the problem X.

Given the the message-passing model $\mathcal{AMP}_{n,n-1}[\emptyset]$, a problem A is *stronger* than a problem B (denoted $A \succeq B$) if B can be solved in $\mathcal{AMP}_{n,n-1}[A]$ (we also say that B is *weaker* than A, denoted $B \preceq A$). Moreover, A is *strictly stronger* than B (denoted $A \succ B$) if $A \succeq B$ and $\neg(B \succeq A)$ (A cannot be solved in $\mathcal{AMP}_{n,n-1}[B]$).

Lemma 3 $P\big(\{k_1, ..., k_s\}, \{k'_1, ..., k'_{s'}\}\big) \Rightarrow \big(\{k_1, \ldots, k_s\}\text{-SSA} \succeq \{k'_1, \ldots, k'_{s'}\}\text{-SSA}\big)$.

Proof Let us consider an algorithm $\mathcal{A}$ that solves the $\{k_1, \ldots, k_s\}$-SSA problem. To solve the $\{k'_1, \ldots, k'_{s'}\}$-SSA problem it is sufficient that each process p_i executes $\mathcal{A}$ until it decides a pair (x, v) and then outputs the pair $(f(x), v)$ as the decided value for the $\{k'_1, \ldots, k'_{s'}\}$-SSA problem (where f is the function appearing in P).

The validity and termination properties of the obtained algorithm follow directly from those of $\mathcal{A}$. Moreover, according to the agreement property satisfied by $\mathcal{A}$, at most k_x different pairs $(x, -)$ can be decided by the underlying algorithm. It then follows from the definition of f that for any $y \in \{1, \ldots, s'\}$, at most $\sum_{x \in f^{-1}(y)} k_x = k_y$ distinct pairs $(y, -)$ are decided in the simulated solution of the $\{k'_1, \ldots, k'_{s'}\}$-SSA problem, and the agreement property of the $\{k'_1, \ldots, k'_{s'}\}$-SSA is satisfied. $\square_{Lemma\ 3}$

Lemma 4 *Let $n > K \geq 2$.* $\big(\{k_1, \ldots, k_s\}\text{-SSA} \succeq \{k'_1, \ldots, k'_{s'}\}\text{-SSA}\big) \Rightarrow P\big(\{k_1, \ldots, k_s\}, \{k'_1, \ldots, k'_{s'}\}\big)$.

Proof The principle and structure of the proof are as follows.

- The proof considers first the failure detector $GZ_{k_1,...,k_s}$, which is sufficient to solve the $\{k_1, ..., k_s\}$-SSA problem (the (s, k)-SSA algorithm described in Section 3 can be easily adapted to solve the $\{k_1, ..., k_s\}$-SSA problem in $\mathcal{AMP}_{n,n-1}[GZ_{k_1,...,k_s}]$).
- As $\{k_1, \ldots, k_s\}$-SSA $\succeq \{k'_1, \ldots, k'_s\}$-SSA, it is possible to solve the $\{k'_1, \ldots, k'_{s'}\}$-SSA problem in $\mathcal{AMP}_{n,n-1}[GZ_{k_1,...,k_s}]$. Moreover, as $GZ(Q)_{k'_1,...,k'_{s'}}$ is necessary to solve the $\{k'_1, \ldots, k'_{s'}\}$-SSA problem (Theorem 4), it is possible to simulate $GZ(Q)_{k'_1,...,k'_{s'}}$ in $\mathcal{AMP}_{n,n-1}[GZ_{k_1,...,k_s}]$.
- The proof constructs such a simulation and shows that this simulation allows to define a function f such that the predicate $P\big(\{k_1, \ldots, k_s\}, \{k'_1, \ldots, k'_{s'}\}\big)$ is satisfied, from which the lemma follows.

Assuming $n > K \geq 2$, and $\mathcal{A}$ being an algorithm that simulates $GZ(Q)_{k'_1,...,k'_{s'}}$ in $\mathcal{AMP}_{n,n-1}[GZ_{k_1,...,k_s}]$, let $QR_i[1, \ldots, s']$ denote the outputs of the simulated quorum at process p_i. Moreover, for each process $p_i, 1 \leq i \leq K+1$ and for each $x \in \{1, \ldots, s\}$, let α_i^x be an execution of $\mathcal{A}$ in which the only process to take steps is p_i and the underlying failure detector $GZ_{k_1,...,k_s}$ always outputs (locally at each p_i) $qr_i[x] = \{i\}$, $ld_i[x] = i$, and $\forall x' \neq x, qr_i[x'] = \{1, \ldots, K+1\}$ and $ld_i[x'] = i$ (let us notice that these outputs comply with the definition of the underlying failure detector $GZ_{k_1,...,k_s}$). According to the quorum liveness property of the constructed failure detector $GZ(Q)_{k'_1,...,k'_{s'}}$, it exists in α_i^x an instant τ_i^x and an index c_i^x such that $\forall \tau \geq \tau_i^x \ : \ QR_i[c_i^x] = \{i\}$.

Considering the solo executions $\{\alpha_i^x\}_{1 \leq i \leq K+1}$, one can build, for any subset of K distinct processes such that $\{i_1, \ldots, i_K\} \subset \{1, \ldots, K+1\}$, an execution of $\mathcal{A}$ denoted

$$\alpha_{i_1 \cdots i_K} = \alpha^1_{i_1} \cdot \alpha^1_{i_2} \cdots \alpha^1_{i_{k_1}}, \alpha^2_{i_{k_1+1}} \cdots \alpha^2_{i_{k_1+k_2}}, \ldots, \alpha^s_{i_{K-k_s+1}} \cdots \alpha^{\cdot}_{i_K} s$$

as follows where $W(x)$ denotes the integer interval $[1 + k_1 + \cdots + k_{x-1}..k_1 + \cdots + k_x]$:

- All processes $p_i, i > K+1$, crash before taking any step,
- All message receptions are delayed after time $\tau\_max = \max\{\tau_i^x$ such that $i \in \{1, ..., K+1\}, x \in \{1, ..., s\}\}$,[2]

[2]This is the only place where the "asynchronous message-passing" assumption is used. If communication was through atomic read/write registers, this message asynchrony would not exist and the lemma would not hold.

- Let us observe that $\alpha_{i_1 \cdots i_K}$ contains the prefix of the solo execution $\alpha^x_{i_w}$ of the process p_{i_w} if and only if $w \in W(x)$.

  For each $x \in \{1, \ldots, s\}$ and for each $w \in W(x)$, the outputs at p_{i_w} of the underlying failure detector $GZ_{k_1,\ldots,k_s}$ are the same as in $\alpha^x_{i_w}$ until τ_max. After this instant, for each $i \in \{i_1, \ldots, i_K\}$ and for $x \in \{1, \ldots, s\}$, $qr_i[x] = \{i_1, \ldots, i_K\}$ and $ld_i[x] = i_1$.

Let us remark that the outputs of the underlying failure detector $GZ_{k_1,\ldots,k_s}$ are valid in this execution, and that, for each $x \in \{1, \ldots, s\}$ and each $p_{i_w}, w \in W(x)$, the execution $\alpha_{i_1 \cdots i_K}$ is indistinguishable from $\alpha^x_{i_w}$ until τ_max from the point of view of p_{i_w}. It follows that for each $x \in \{1, \ldots, s\}$ and each $p_{i_w}, w \in W(x)$, we have $QR_{i_w}[c^x_{i_w}] = \{i_w\}$ at time τ_max in $\alpha_{i_1 \cdots i_K}$. It follows from this observation and the quorum intersection property of the failure detector $GZ(Q)_{k'_1,\ldots,k'_{s'}}$ built by $\mathcal{A}$ (this property has to be preserved on each quorum index $y \in \{1, \ldots, s'\}$) that $\{c^1_{i_1}, \ldots, c^1_{i_{k_1}}, \ldots, c^s_{i_{K-k_s+1}}, \ldots, c^s_{i_K}\}$ is a multiset of K elements where for each $y \in \{1, \ldots, s'\}$, the value y appears k'_y times (and those are the only values of the elements of this multiset).

Let us now consider the execution $\alpha_{\sigma(1)\cdots\sigma(K)}$ where σ is any permutation of $\{1, \ldots, K\}$. According to the previous discussion, any process $p_{i_{\sigma(w)}}, w \in \{1, \ldots, K\}$ can be replaced by p_{K+1} in the execution without changing the multiset $\{c^1_{i_\sigma(1)}, \ldots, c^1_{i_{\sigma(k_1)}}, \ldots, c^s_{i_{\sigma(K-k_s+1)}}, \ldots, c^s_{i_{\sigma(K)}}\}$. It then follows that $\forall x \in \{1, \ldots, s\}, \forall w \in \{1, \ldots, K\} : (\sigma(w) \in W(x) \Rightarrow c^x_{i_{\sigma(w)}} = c^x_{K+1})$. As σ can be any permutation, it follows that $\forall x \in \{1, \ldots, s\}, \forall w \in \{1, \ldots, K\} : c^x_w = c^x_{K+1}$. Moreover this implies that $\forall x \in \{1, \ldots, s\}, \forall i, j \in \{1, \ldots, K+1\} : c^x_i = c^x_j$.

Let us finally consider the function f which, with each $x \in \{1, \ldots, s\}$ associates c^x_1 (which, as just shown, is equal to c^x_i for all $i \in \{1, \ldots, K\}$). Since, $\{c^1_{i_1}, \ldots, c^1_{i_{k_1}}, \ldots, c^s_{i_{K-k_s+1}}, \ldots, c^s_{i_K}\}$ contains k_x times $f(x)$ for each $x \in \{1, \ldots, s\}$ (and only these values), it follows from the multiset equality above that $\forall y \in \{1, \ldots, s'\} : \Sigma_{x \in f^{-1}(y)} k_x = k_y$, which ends the proof of the lemma. $\square_{Lemma\ 4}$

Theorem 5 *Let $n > K \geq 2$.* $\big(\{k_1, \ldots, k_s\}\text{-SSA} \succeq \{k'_1, \ldots, k'_{s'}\}\text{-SSA}\big) \Leftrightarrow P\big(\{k_1, \ldots, k_s\}, \{k'_1, \ldots, k'_{s'}\}\big)$.

Proof The proof follows directly from Lemma 3 and Lemma 4. $\square_{Theorem\ 5}$

Theorem 6 *The relation $\succ$ on generalized-*SSA *problems is a partial order.*

Proof The proof follows from Theorem 3 ($G(K)$ is a partial order), Lemma 2 (all paths in ($G(K)$ are characterized by P), and Theorem 5 (which relates P and $\succ$). $\square_{Theorem\ 6}$

The next corollary follows from the observation that, for any $K > 1$, the K-set agreement problem is a sink vertex in the directed graph $G(K)$.

Corollary 1 *The weakest failure detector for the K-set agreement problem does not allow to solve any $\{k_1, \ldots, k_s\}$-SSA problem such that $s > 1$ and $k_1 + \cdots + k_s = K$.*

5.5 The Lattice of Symmetric SSA Problems

As seen before, a symmetric vertex is a vertex $\{k_1, ..., k_s\}$ such that $k_1 = \ldots = k_s = k$. Let $SG(K)$ denote the graph whose vertices are the symmetric vertices of $G(K)$, and there is an edge from (s_x, k_x) to (s_y, k_y) iff there is a path in $G(K)$ from the vertex $\{k_x, ..., k_x\}$ (k_x appearing s_x times) to the vertex $\{k_y, ..., k_y\}$ (k_y appearing s_y times) and no path connecting these vertices passes through a symmetric vertex. As an example, $SG(6)$ is given in Section 5.2.

Theorem 7 *For any K, $SG(K)$ is a lattice.*

Proof Let (s_x, k_x) and (s_y, k_y) be two pairs of integers such that $k_x < k_y$ and $s_x k_x = s_y k_y = K$. It follows from the definition of $SG(K)$ that there is an edge from (s_x, k_x) to (s_y, k_y) iff $k_y = k_x p$ where p is prime (if p is not prime there is a symmetric vertex on a path from (s_x, k_x) to (s_y, k_y), and if k_y/k_x is not an integer, there is no path from (s_x, k_x) to (s_y, k_y)). It follows that we can associate with any pair of pairs (s_x, k_x) and (s_y, k_y) such that $s_x k_x = s_y k_y = K$,

- among its ancestors, the vertex (s_z, k_z) in $SG(K)$ where $s_z k_z = K$ and $k_z = gcd(k_x, k_y)$, and
- among its successors, the vertex (s'_z, k'_z) in $SG(K)$ where $s'_z k'_z = K$ and $k'_z = lcm(k_x, k_y)$.

As the greatest common denominator and the least common multiple of any pair of integers are unique, it follows that $SG(K)$ is a lattice. $\square_{Theorem\ 7}$

The next corollary follows from the previous theorem.

Corollary 2 *Let (s_1, k_1) and (s_2, k_2) be two different pairs of integers such that $s_1 k_1 = s_2 k_2$, and none of k_1 and k_2 divides the other one. The symmetric (s_1, k_1)-SSA and (s_2, k_2)-SSA problems are incomparable in $\mathcal{AMP}_{n,n-1}[\emptyset]$.*

As far as agreement problems are concerned, this shows a strong difference between the message-passing model and the read/write model. In the read/write model, (s_1, k_1)-SSA and (s_2, k_2)-SSA are the same problem (they are both equivalent to the K-simultaneous problem which is itself equivalent to the K-set agreement problem, where $K = s_1 k_1 = s_2 k_2$).

6 Conclusion

This paper has investigated the comparative power of simultaneous agreement and set agreement in asynchronous message-passing systems prone to any number of process crashes. This study was initially motivated by the lasting (and difficult) quest for the weakest failure detector for the k-set agreement problem in message-passing systems.

While k-simultaneous consensus and k-set agreement are equivalent problems in asynchronous read/write systems prone to any number of process crashes [1], the paper has introduced a general formulation of agreement problems, namely the family of (s, k)-SSA (s-simultaneous k-set agreement) problems, and has shown that these agreement problems define a strong hierarchy, thereby showing that their shared memory equivalence is no longer true in message-passing systems. Hence, this study contributes to a better understanding of the relation (equivalence/difference) between the send/receive model and the read/write model (equivalence when a majority of processes are correct [2], and difference –from a problem ranking point of view– in the other cases).

Acknowledgments

This work has been partially supported by the French ANR project DISPLEXITY devoted to computability and complexity in distributed computing.

References

[1] Afek Y., Gafni E., Rajsbaum S., Raynal M. and Travers C., The k-Simultaneous Consensus Problem. *Distributed Computing*, 22:185-195, 2010.

[2] Attiya H., Bar-Noy A. and Dolev D., Sharing Memory Robustly in Message Passing Systems. *Journal of the ACM*, 42(1):121-132, 1995.

[3] Bonnet F. and Raynal M., On the Road to the Weakest Failure Detector for k-Set Agreement in Message-passing Systems. *Theoretical Computer Science*, 412(33):4273-4284, 2011.

[4] Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on Theory of Computation (STOC'93)*, pp. 91-100, 1993.

[5] Bouzid Z. and Travers C., (Anti-$\Omega_k \times \Sigma_k$)-Based k-Set Agreement Algorithms. *Proc. 12th Int'l Conference on Principles of Distributed Systems (OPODIS'10)*, Springer LNCS 6490, pp. 190-205, 2010.

[6] Bouzid Z. and Travers C., Simultaneous Consensus is Harder than Set Agreement in Message-Passing. http://hal.inria.fr/hal-00752610, 2012.

[7] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, 1996.

[8] Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[9] Chaudhuri S., More *Choices* Allow More *Faults:* Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation,* 105:132-158, 1993.

[10] Delporte-Gallet C., Fauconnier H. and Guerraoui R., Tight Failure Detection Bounds on Atomic Object Implementations. *Journal of the ACM*, 57(4):Article 22, 2010.

[11] Delporte-Gallet C., Fauconnier H., Guerraoui R. and Tielmann A., The Weakest Failure Detector for Message Passing Set-Agreement. *Proc. 22th Int'l Symposium on Distributed Computing (DISC'08)*, Springer LNCS 5218, pp. 109-120, 2008.

[12] Gafni E. and Kuznetsov P., On Set Consensus Numbers. *Distributed Computing*, 24(3-4):149-163, 2011.

[13] Guerraoui R. and Raynal M., The Alpha of Indulgent Consensus. *The Computer Journal*, 50(1):53-67, 2007.

[14] Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923, 1999.

[15] Mostéfaoui A. and Raynal M., k-Set Agreement with Limited Accuracy Failure Detectors. *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 143-152, 2000.

[16] Mostéfaoui A., Raynal M. and Stainer J., Relations Linking Failure Detectors Associated with k-Set Agreement in Message-Passing Systems. *Proc. 13th Int'l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'11)*, Springer LNCS 6976, pp. 341-355, 2011.

[17] Mostéfaoui A., Raynal M. and Stainer J., Chasing the Weakest Failure Detector for k-Set Agreement in Message-Passing Systems. *Proc. 11th IEEE Int'l Symposium on Network Computing and Applications (NCA'12)*, IEEE Press, pp. 44-51, 2012.

[18] Raynal M., K-anti-Omega. *Rump Session at 26th ACM Symposium on Principles of Distributed Computing (PODC'07)*, 2007.

[19] Raynal M., *Fault-Tolerant Agreement in Synchronous Message-Passing Systems*. Morgan & Claypool Publishers, 165 pages, 2010 (ISBN 978-1-60845-525-6).

[20] Raynal M., Failure Detectors to Solve Asynchronous k-Set Agreement: a Glimpse of Recent Results. *Bulletin of the EATCS*, 103:74-95, 2011.

[21] Raynal M., *Concurrent Programming: Algorithms, Principles, and Foundations*. Springer, 515 pages, 2013 (ISBN 978-3-642-32027-9).

[22] Raynal M. and Travers C., In Search of the Holy Grail: Looking for the Weakest Failure Detector for Wait-free Set Agreement. *Proc. 10th Int'l Conf. on Principles of Distributed Systems (OPODIS'06)*, Springer LNCS 4305, pp. 3-19, 2006.

[23] Saks M. and Zaharoglou F., Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.

[24] Zielinski P., Anti-Omega: the Weakest Failure Detector for Set Agreement. *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*, ACM Press, pp. 55-64, 2008.