Conditions for Set Agreement with an Application to Synchronous Systems

François Bonnet, Michel Raynal

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1870

CONDITIONS FOR SET AGREEMENT WITH AN APPLICATION TO SYNCHRONOUS SYSTEMS

F. BONNET M. RAYNAL

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

Conditions for Set Agreement with an Application to Synchronous Systems

F. Bonnet* M. Raynal**

Systèmes communicants
Projet ASAP

Publication interne n ° 1870 — Novembre 2007 — 21 pages

Abstract: The k-set agreement problem is a generalization of the consensus problem: considering a system made up of n processes where each process proposes a value, each non-faulty process has to decide a value such that a decided value is a proposed value, and no more than k different values are decided. While this problem cannot be solved in an asynchronous system prone to t process crashes when $t \geq k$, it can always be solved in a synchronous system; $\lfloor \frac{t}{k} \rfloor + 1$ is then a lower bound on the number of rounds (consecutive communication steps) for the non-faulty processes to decide. The *condition-based* approach has been introduced in the consensus context. Its aim was to both circumvent the consensus impossibility in asynchronous systems, and allow for more efficient consensus algorithms in synchronous systems. This paper addresses the condition-based approach in the context of the k-set agreement problem. It has two main contributions.

The first is the definition of a framework that allows defining conditions suited to the ℓ-set agreement problem. More precisely, a condition is defined as a set of input vectors such that each of its input vectors can be seen as "encoding" ℓ values, namely, the values that can be decided from that vector. A condition is characterized by the parameters t, ℓ, and a parameter denoted d such that the greater $d + \ell$, the least constraining the condition (i.e., it includes more and more input vectors when $d + \ell$ increases, and there is a condition that includes all the input vectors when $d + \ell > t$). The conditions characterized by the triple of parameters t, d and ℓ define the class of conditions denoted $\mathcal{S}_t^{d,\ell}$, $0 \leq d \leq t$, $1 \leq \ell \leq n - 1$. The properties of the sets $\mathcal{S}_t^{d,\ell}$ are investigated, and it is shown that they have a lattice structure.

The second contribution is a generic synchronous k-set agreement algorithm based on a condition $C \in \mathcal{S}_t^{d,\ell}$, i.e., a condition suited to the ℓ-set agreement problem, for $\ell \leq k$. This algorithm requires at most $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$ rounds when the input vector belongs to C, and $\lfloor \frac{t}{k} \rfloor + 1$ rounds otherwise. (Interestingly, this algorithm includes as particular cases the classical synchronous k-set agreement algorithm that requires $\lfloor \frac{t}{k} \rfloor + 1$ rounds (case $d = t$ and $\ell = 1$), and the synchronous consensus condition-based algorithm that terminates in $d + 1$ rounds when the input vector belongs to the condition, and in $t + 1$ rounds otherwise (case $k = \ell = 1$).)

Key-words: Agreement problem, Condition, Crash failure, Distributed algorithm, Efficiency, Fault-tolerance, Input vector, Lower bound, Round-based computation, Set agreement, Synchronous system.

(Résumé : tsvp)

* IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France Francois.Bonnet@irisa.fr
** IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France raynal@irisa.fr

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Conditions pour l’accord ensembliste

Résumé : Ce rapport présente des conditions adaptées à l’accord ensembliste et un protocole synchrone fondé sur ces conditions.

Mots clés : Accord ensembliste, condition.

1 Introduction

1.1 Context of the paper

The consensus problem and the condition-based approach The condition-based approach has been introduced to circumvent the impossibility of solving the consensus problem in asynchronous systems prone to process crashes. That problem can be stated as follows. Each process proposes a value, and the processes have to cooperate in such a way that each non-faulty process decides a value (termination), a decided value is a proposed value (validity), and no two processes decide different values (agreement). Consensus is a fundamental problem that lies at the core of nearly all the distributed coordination or consistency problems encountered in fault-tolerant distributed computing. Despite its very simple definition, it is surprising that this problem cannot be solved in asynchronous systems prone to process crashes (even in presence of a single crash), be the underlying communication medium a reliable message passing system [10] or a shared memory made up of read/write registers [17].

Given a problem, the condition-based approach analyzes restrictions of the problem to subsets of its inputs [20]. Each restriction defines a new problem that is a particular instance of the original problem. Conditions have given rise to two lines of research, one focused on decidability, the other on efficiency. More precisely, it addresses the two following questions.

- Given an unsolvable problem (e.g., asynchronous consensus), for what restrictions on its inputs does the problem become solvable?
- Given a solvable problem (e.g., synchronous consensus), for what restrictions on its inputs does the problem become easier to solve (i.e., solvable in a more efficient way)?

Up to now, nearly all the results of the condition-based approach concerns the consensus problem (see e.g., [11, 14, 15, 20, 22, 24, 29]). Let an input vector be a vector with one entry per process, the entry associated with a given process containing the value proposed by that process. Each input vector can be seen as a codeword encoding one of its values, namely, the value decided by the non-faulty processes [11]. A condition is a set of input vectors (codewords). In this context, the asynchronous consensus impossibility can be restated as follows: no condition can contain all the possible input vectors. So, one of the most fundamental question of the condition-based approach is the characterization of the largest set of conditions that allows to solve the consensus problem in a crash-prone asynchronous system. This question has been answered in [20] where is introduced the notion of x-legal condition and is proved the following characterization: a condition C allows to solve consensus in an asynchronous system prone to up to x process crashes iff it is x-legal [20] (a condition is x-legal if each of its input vectors contains the same value more than x times, and the Hamming distance of two vectors from which different values can be decided is greater than x).

Condition-based synchronous consensus The consensus problem can be solved in a synchronous system prone to any number of process crashes. In such systems, the time efficiency of an algorithm is measured in terms of the number R of rounds (consecutive communication steps) needed for the non-faulty processes to decide. It has been shown that $t+1$ is a lower bound for R where t is an upper bound on the number of faulty processes [2, 9].

As shown in [22, 29], the condition-based approach allows bypassing that lower bound each time the input vector belongs to the condition. Families of conditions, denoted $\left(\mathcal{S}_t^d\right)_{0\leq d\leq t}$, are introduced in [22]; the parameter d is called the *degree* of the condition (and the quantity $(t-d)$ measures the *difficulty* of the condition). A condition C belongs to $\mathcal{S}_t^d$ if it is $(t-d)$-legal. The following hierarchy of sets of conditions for synchronous consensus has been established in [22]:

$$\mathcal{S}_t^0 \subset \mathcal{S}_t^1 \subset \cdots \subset \mathcal{S}_t^d \subset \cdots \subset \mathcal{S}_t^t.$$

Let us consider a synchronous system where up to t processes can crash, a condition $C \in \mathcal{S}_t^d$ and an input vector I. The main result of [22] is the following. if $I \in C$, consensus can be solved in two rounds when $d=0$, and in $d+1$ rounds when $1 \leq d \leq t$. When $I \notin C$, the number of rounds is $t+1$ rounds (as already known). That paper also proves that $d+1$ is a tight lower bound for R, when the input vector belong to C (with $C \in \mathcal{S}_t^d$ and $C \notin \mathcal{S}_t^{d-1}$).

It is worthwhile looking at the "extreme" sets $\mathcal{S}_t^0$ and $\mathcal{S}_t^t$. One one side, $\mathcal{S}_t^t$ includes the condition that contains all the possible input vectors. On the other side, the family of conditions $\mathcal{S}_t^0$, that is the largest set of conditions that allow *to solve* the consensus problem in asynchronous systems prone to up to t crashes, is also the family of conditions that allows to solve the consensus problem *optimally* in a synchronous system prone to t crashes. As far as consensus is concerned, this establishes a *simple* and *well-defined* borderline relating efficiency in synchronous systems and computability in asynchronous systems.

Set agreement The k-set agreement problem has been introduced to investigate how the number of choices (k) allowed to the processes is related to the maximum number (t) of processes that can crash [6]. More precisely, the processes are allowed to decide up to k different values (consensus is 1-set agreement). While that problem can be trivially solved in asynchronous systems where $k > t$, it has no solution in these systems as soon as $k \leq t$ [5, 13, 28]. The situation is different in synchronous systems where the k-set agreement problem can always be solved, whatever the values of t and k. In these systems, the lower bound on the number of rounds is $R = \lfloor \frac{t}{k} \rfloor + 1$ [7].

Few works have considered the condition-based approach for solving the k-set agreement problem. A topology-based characterization of the conditions that allow to wait-free solve the $(n-1)$-set agreement problem is presented in [3] (wait-free means that the only value of t that is considered is $t = n-1$). Condition-based asynchronous shared memory k-set agreement algorithms are presented in [23] (without characterizing conditions suited to k-set agreement). An asynchronous shared memory algorithm is presented in [21] [1]. Assuming that the input vector I belongs to a condition $C \in \mathcal{S}_t^d$, that algorithm solves the k-set agreement problem for $k = d+1$. That algorithm can trivially be transformed in a one-round synchronous algorithm. (It is also shown in [21] that, for $k \leq d$, there is no k-set agreement algorithm when $C \in \mathcal{S}_t^d$ and $C \notin \mathcal{S}_t^{d-1}$.)

1.2 Motivation and content of the paper

This paper is on the condition-based approach for solving the k-set agreement problem. It originates from the following observations and associated questions.

A question on the dividing power of conditions in synchronous systems Let us assume that the input vector I belongs to a condition $C \in \mathcal{S}_t^d$ (i.e., C allows to solve consensus in an asynchronous system in which up to $(t-d)$ processes may crash). Moreover, let us consider the pair (k, R) where R is the number of rounds to solve synchronous k-set agreement despite up to t crashes.

When looking at the results described in [21, 22], we have two synchronous algorithms: (i) one solves consensus in $d+1$ rounds [22], i.e., it realizes the pair $(1, d+1)$; (ii) the other solves $(d+1)$-set agreement in one round [21], i.e., it realizes the pair $(d+1, 1)$. This observation gives rise to the following questions: Are there algorithms for other (k, R) pairs? If yes, what is the generic formulation of the (k, R) pair, i.e., what is the relation linking R, k and d?

This issue was the initial question this paper originated from. The paper answers it by presenting a condition-based algorithm for the generic pair $(k, \lfloor \frac{d}{k} \rfloor + 1)$. This means that if the algorithm stops after r rounds, the processes decide on at most k values where k is the smallest integer such that $\lfloor \frac{d}{k} \rfloor + 1 \leq r$.

Interestingly, this generalizes to conditions the fact that, when we go from synchronous consensus to synchronous k-set agreement, the time complexity is always divided by k. This was not a priori evident, as the bound $\lfloor \frac{t}{k} \rfloor + 1$ is purely "syntactic" (it is only based on the worst failure pattern), while the condition-based approach is more "semantic" (it is based on the actual input values).

From smaller conditions for consensus to larger conditions for set agreement The set of conditions $\left(\mathcal{S}_t^d\right)_{0 \leq d \leq t}$ have been designed with the consensus problem in mind. This means that, for any $C \in \mathcal{S}_t^d$, each input vector $I \in C$ is seen as "encoding" a single value, namely, the value decided when the input vector is I. As we have just seen [21, 22], such a *consensus condition* C can be used to solve the k-set agreement with a round complexity ($\lfloor \frac{d}{k} \rfloor + 1$) smaller than the one required for solving consensus ($d+1$). This can be seen as a simple "side effect" of the fact that the conditions of $\mathcal{S}_t^d$ are defined for solving the consensus problem. It is not at all counter-intuitive that using a condition designed for a *stronger* problem (consensus) can allow for more efficient solutions when used to solve *weaker* problems (k-set agreement for $k > 1$).

This observation suggests the following question. Considering the ℓ-set agreement problem ($\ell \geq 1$) in a synchronous system prone to t process crashes, is it possible to design conditions directly suited to that problem, i.e., devise families of conditions $\mathcal{S}_t^{d,\ell}$, $0 \leq d \leq t$, such that there are conditions $C \in \mathcal{S}_t^{d,\ell}$ that allow to solve efficiently the ℓ-set agreement problem while they do not necessarily allow to solve efficiently $(\ell-1)$-set agreement? [2] Said

[1] That algorithm actually combines a condition $C \in \mathcal{S}_t^d$ and a failure detector of a class denoted ψ_t^y. It solves k-set agreement with $k = 1 + \max(0, d-y)$. Here we consider the case where the failure detector offers no additional power, i.e., $y = 0$.

[2] As a simple example, the condition C that contains all the vectors whose entries contain exactly two different values allows to solve very efficiently the 2-set agreement problem despite any number of process crashes, while it does not allow to solve efficiently consensus in the same failure context.

differently, can an input vector be seen as encoding up to ℓ different values? (Taking $\ell = 1$ boils down to a condition in $C \in \mathcal{S}_t^d = \mathcal{S}_t^{d,1}$.)

The paper presents and investigates such a family of conditions $\mathcal{S}_t^{d,\ell}$, $0 \leq d \leq t$. As we will see in the paper, when $\ell > t - d$, $\mathcal{S}_t^{d,\ell}$ contains the trivial condition including all the possible input vectors. More generally, the paper establishes the following hierarchy for the synchronous ℓ-set agreement problem:

$$\mathcal{S}_t^{0,\ell} \subset \mathcal{S}_t^{1,\ell} \subset \cdots \subset \mathcal{S}_t^{d,\ell} \subset \cdots \subset \mathcal{S}_t^{t+1-\ell,\ell} = \mathcal{S}_t^{t+2-\ell,\ell} = \cdots = \mathcal{S}_t^{t,\ell}.$$

Given such a hierarchy, an important question is the following one. Considering a condition $C \in \mathcal{S}_t^{d,\ell}$ and assuming $\ell \leq k$ (otherwise, a condition $C \in \mathcal{S}_t^{d,\ell}$ is useless to solve k-set agreement), how can C help solve the k-set agreement problem in a synchronous system prone to t crashes? The paper answers this question by presenting an algorithm that, when the input vector belongs to the condition $C \in \mathcal{S}_t^{d,\ell}$ the algorithm is instantiated with, realizes the pair (k, R) where

$$R = \left\lfloor \frac{d - 1 + \ell}{k} \right\rfloor + 1.$$

When the input vector does not belong to C, the algorithm requires at most $\lfloor \frac{t}{k} \rfloor + 1$ rounds. Given t and ℓ, this means that, when we consider the previous hierarchy $\left(\mathcal{S}_t^{d,\ell}\right)_{0 \leq d \leq t+1-\ell}$, there are less and less conditions when d decreases (and these conditions contain less and less vectors), but these conditions always allow for a faster decision each time the input vector belongs to the condition.

This noteworthy formula pieces together all the relevant parameters. It involves, on one side, the coordination degree k associated with the set agreement problem we want to solve (i.e., the difficulty of the problem instance we want to solve, which increases when k decreases), and, on the other side, the pair (d, ℓ) characterizing the condition the input vector belongs to (i.e., the help we receive to solve the problem, which increases when d or ℓ decreases).

As a simple example, let us consider the case where $d = t + 1 - \ell$, i.e., the set $\mathcal{S}_t^{t+1-\ell,\ell}$. As previously indicated, that set includes the condition containing all the possible input vectors. We consequently obtain $R = \lfloor \frac{t}{k} \rfloor + 1$, the classical lower bound when no condition is used [7].

1.3 Roadmap

The paper is made up of 8 sections. Section 2 defines the notion of (x, ℓ)-legality (the conditions in $\mathcal{S}_t^{d,\ell}$ are $(t - d, \ell)$-legal). Section 3 investigates the structure of the sets of (x, ℓ)-legal conditions for $0 \leq x < n$ and $0 < \ell \leq n$. More precisely, for each pair of pairs, (x_1, ℓ_1) and (x_2, ℓ_2), that section states if an (x_1, ℓ_1)-legal condition is also an (x_2, ℓ_2)-legal condition or not. Interestingly, this structure shows also the following: the condition including all the input vectors is (x, ℓ)-legal only if $\ell > x$, and, for $\ell > x$, the set of (x, ℓ)-legal conditions contains the condition including all input vectors[3]. Section 4 briefly considers (x, ℓ)-legality in asynchronous systems, while Section 5 presents hierarchies of sets of conditions suited to synchronous ℓ-set agreement. Then, Section 6 presents a synchronous k-set agreement algorithm whose properties have been previously described. Section 7 proves its correction. Finally, Section 8 provides concluding remarks and presents open problems.

2 Conditions for set agreement

This section and the following (on the structure of the set of conditions) are independent of the underlying synchrony assumption and the way processes communicate (message-passing vs shared memory).

2.1 Preliminaries

Process Model The system consists of a finite set of n processes denoted $\Pi = \{p_1, \ldots, p_n\}$. A process is *faulty* during an execution if it prematurely stops its execution (crash). After it has crashed, a process does nothing. A *correct* process is a process that is not faulty. As already mentioned, t denotes the upper bound on the number of faulty processes ($1 \leq t < n$).

[3] Intuitively, this seems to be related to the impossibility to solve the ℓ-set agreement problem in an asynchronous system prone to x process crashes when $\ell \leq x$. Impossibility means here that there is no asynchronous algorithm when the actual input vector can be any input vector.

The set agreement problem The k-set agreement problem has been informally stated in the Introduction: every process p_i *proposes* a value v_i and all correct processes have to *decide* on a value v, in relation to the set of proposed values. More precisely, the problem is defined by the following three properties:

- Termination. Every correct process eventually decides.
- Validity. If a process decides v, then v was proposed by some process.
- Agreement. At most k different values are decided.

Notation In the following $\mathcal{V}$ denotes the set of values that can be proposed by the processes, and $\perp$ denotes a default value that cannot be proposed by the processes.

An *input vector* I is a vector of size n (denoted $|I| = n$), whose i-th entry contains the value proposed by p_i, or $\perp$ if p_i did not take any step in the execution, where $\perp$ denotes a default value such that $\perp \notin \mathcal{V}$ and $\forall a \in \mathcal{V}, \perp < a$. We usually denote by I a vector with all entries in $\mathcal{V}$, and with J a vector that may have some entries equal to $\perp$; such a vector J is called a *view*.

The values of $\mathcal{V}$ that are present in I defines a subset of $\mathcal{V}$ denoted $val(I)$; $|val(I)|$ denotes its cardinality.

Let $J1, J2$ with some entries are possibly equal to $\perp$. We say "$J2$ *contains* $J1$" (denoted $J1 \leq J2$) if $\forall k : J1[k] \neq \perp \Rightarrow J1[k] = J2[k]$. The number of occurrences of a value a in the vector J, with $a \in \mathcal{V} \cup \{\perp\}$ is denoted $\#_a(J)$.

$d_H(J_1, J_2)$ denotes the Hamming distance separating J_1 and J_2, i.e., the number of entries in which J_1 and J_2 differ. Moreover, given a non-empty set of vectors $\{J_1, \ldots, J_z\}$, $d_G(J_1, \ldots, J_z)$ denotes the number of distinct entries for which at least two vectors in $J_1, \ldots, J_z$ differ. We call $d_G(J_1, \ldots, J_z)$ the *generalized* distance of the set of vectors $J_1, J_2, \ldots, J_z$. As an example, $d_G([a, \perp, a, e, b, b], [a, \perp, a, e, c, c], [a, \perp, f, e, b, c]) = 3$. As we can see, when $d_G()$ is on two vectors, it boils down to the Hamming distance.

Let $I_1, \ldots, I_z$ be a set of vectors. $\cap_{1 \leq j \leq z} I_j$ denotes the vector containing the $n - d_G(I_1, \ldots, I_z)$ entries that belong to all the vectors $I_1, \ldots, I_z$. It is called the *intersecting* vector of $I_1, \ldots, I_z$. When clear from the context, the notation $\cap_{1 \leq j \leq z} I_j$ is also used to denote the set of values in that intersecting vector.

2.2 The notion of (x, ℓ)-legality

Definition 1 *A condition is a set of input vectors.*

Definition 2 *Let $0 < \ell \leq n$. A condition C is (x, ℓ)-legal if there is a function $h_\ell()$ that satisfies the following properties:*

1. (x, ℓ)-Validity. $\forall I \in C : h_\ell(I) \subseteq val(I)$ *and* $|h_\ell(I)| = \min(\ell, |val(I)|)$.
2. (x, ℓ)-Density. $\forall I \in C : \Sigma_{v \in h_\ell(I)} \#_v(I) > x$,
3. (x, ℓ)-Distance. $\forall \alpha : 0 \leq \alpha < x, \forall I_1, \ldots, I_z \in C$:
   $d_G(I_1, \ldots, I_z) = x - \alpha \Rightarrow \Sigma_{v \in \bigcap_{1 \leq j \leq z} h_\ell(I_j)} \#_v(\cap_{1 \leq j \leq z} I_j) > \alpha$.

As already indicated, intuitively, an input vector I of an (x, ℓ)-legal condition can be seen as a codeword encoding an "abstract" value. That "abstract" value can be instantiated by any value of a set of ℓ "concrete" values, namely, the values defined by $h_\ell(I)$. The aim of the validity, density and distance properties is to ensure that the function $h_\ell()$ provides a correct "decoding".

The validity property states that at most ℓ values can be decided and those are values that belong to the input vector. The density property guarantees that a decided value can be extracted from an input vector, despite up to x crashes (from an operational point of view, the corresponding entries in the input vector can possibly remain forever equal to $\perp$).

Finally, the distance property guarantees that if a set of input vectors differ in some number of entries (namely, $x - \alpha$), then they must contain values that (1) can be decided from each of them, and (2) are present enough in each of them (namely, more than α times): the intersecting vector $\cap_{1 \leq j \leq z} I_j$ must contain "enough" values of $\bigcap_{1 \leq j \leq z} h_\ell(I_j)$. ([4].)

[4] It could be possible to integrate the density property inside the distance property by considering the case $\alpha = x$. We have not done it, because in some proofs, the case $\alpha = x$ has to be treated separately.

Definition 3 *A function $h_\ell()$ that makes (x,ℓ)-legal a condition C is called an* (x,ℓ)-recognizing *function for C. If additionally the addition to C of any new vector I is such that the condition $C \cup \{I\}$ is not (x,ℓ)-legal, the function $h_\ell()$ is called an* (x,ℓ)-generating *function for C.*

Remark When we consider $\ell = 1$, the previous definition of (x,ℓ)-legality boils down to notion of x-legality introduced in [20]. The distance property simplifies and becomes $h_1(I_1) \neq h_2(I_2) \Rightarrow d_H(I_1, I_2) > x$. It follows that a condition C allows to solve the consensus problem in an asynchronous system prone to x process crashes if and only if it is $(x,1)$-legal.

Theorem 1 *Let C be an (x,ℓ)-legal condition, J a vector such that $\#_\perp(J) \leq x$, and $I_1, \ldots, I_z$ input vectors in C such that $J \leq I_1, \ldots, J \leq I_z$. $0 < |\bigcap_{1\leq j\leq z} h_\ell(I_j) \bigcap val(J)| \leq \ell$.*

Proof Let us first observe that, due to the validity property, we have $\forall\, I_1, \ldots, I_z$: $|h_\ell(I_j)| \leq \ell$, from which it follows that $|\bigcap_{1\leq j\leq z} h_\ell(I_j) \bigcap val(J)| \leq \ell$.

Let us now show that the set $\bigcap_{1\leq j\leq z} h_\ell(I_j) \bigcap val(J)$ is not empty. Let $\#_\perp(J) = x - \beta$ with $0 \leq \beta \leq x$. Let $d_G(I_1, \ldots, I_z) = x - \alpha$. As $J \leq I_1, \ldots, J \leq I_z$, we also have $d_G(I_1, \ldots, I_z) = \#_\perp(J) - \gamma$ with $\gamma \geq 0$. Finally, as $\#_\perp(J) - \gamma = (x - \beta) - \gamma$, we have $x - \alpha = (x - \beta) - \gamma$, i.e., $\beta + \gamma = \alpha$ with $\alpha, \beta, \gamma \geq 0$.

Due either to the density property (case $\alpha = x$) or the distance property (case $0 \leq \alpha < x$), if follows that we have $\Sigma_{v\in\bigcap_{1\leq j\leq z} h_\ell(I_j)} \#_v(\cap_{1\leq j\leq z} I_j) > \alpha$. As $\alpha + 1 = \beta + \gamma + 1 > \gamma$, more than γ entries of this intersecting vector $\cap_{1\leq j\leq z} I_j$ contain values of $\bigcap_{1\leq j\leq z} h_\ell(I_j)$. As each entry of $\bigcap_{1\leq j\leq z} h_\ell(I_j)$ is an entry of J and no entry of $\bigcap_{1\leq j\leq z} h_\ell(I_j)$ is equal to $\perp$, it follows that at least one value in $\bigcap_{1\leq j\leq z} h_\ell(I_j)$ belongs to J. $\square_{Theorem\ 1}$

Thanks to the previous lemma, the function $h_\ell()$ can be extended to vectors J with at most x entries equal to $\perp$.

Definition 4 *Let C be an (x,ℓ)-legal condition, and J a vector such that $\#_\perp(J) \leq x$. Let $I_1, \ldots, I_z$ be the inputs vectors in C (if any) such that $J \leq I_1, \ldots, J \leq I_z$. The definition of $h_\ell()$ is extended to such vectors J as follows: $h_\ell(J) = \bigcap_{1\leq j\leq z} h_\ell(I_j) \bigcap val(J)$. (If there is no input vector I such that $J \leq I$, $h_\ell(J)$ is left undefined.)*

2.3 (x,ℓ)-legality with $h_\ell() = \max_\ell()$

This section investigates the case where $h_\ell(I)$ returns the ℓ greatest values of I. This function is denoted $\max_\ell()$ ($\max_1()$ is denoted $\max()$). Let us observe that all the theorems in this section remain true when the function $\max_\ell()$ is replaced by the function $\min_\ell()$.

Theorem 2 *Let C be the condition the vectors of which satisfy the (x,ℓ)-validity and (x,ℓ)-density properties when considering the function $h_\ell() = \max_\ell()$. The condition C is (x,ℓ) legal.*

Proof Let us consider the condition C that contains all the vectors I that satisfy the (x,ℓ)-size property and the (x,ℓ)-density property, i.e., $I \in C \Rightarrow \Sigma_{v\in\max_\ell(I)} \#_v(I) > x$. Let us observe that C is not empty. So, to prove the theorem, we have to show that any set of vectors $\{I_1, \ldots, I_z\} \subseteq C$ such that $d_G(I_1, \ldots, I_z) = x - \alpha$ (with $0 \leq \alpha < x$) satisfies the (x,ℓ)-distance property, i.e., we have $\Sigma_{v\in\bigcap_{1\leq j\leq z} h_\ell(I_j)} \#_v(\cap_{1\leq j\leq z} I_j) > \alpha$.

Let $D = d_G(I_1, \ldots, I_z)$. Let us consider any vector $I_j \in \{I_1, \ldots, I_z\}$. As $I_j \in C$, we have $\Sigma_{v\in\max_\ell(I_j)} \#_v(I_j) \geq x + 1 = D + \alpha + 1$. In the worst case, (1) the same D entries of the vectors $I_1, \ldots, I_z$ contain only values among their ℓ greatest values, and (2) these entries contain different values. As, for each I_j, $\Sigma_{v\in\max_\ell(I_j)} \#_v(I_j) \geq D + \alpha + 1$, it follows that the vectors $I_1, \ldots, I_z$ share $\alpha + 1$ entries that contain values belonging to their ℓ greatest values. $\square_{Theorem\ 2}$

Size of the condition defined by $h_1() = \max()$ This section computes the number of distinct input vectors that are in the $(x,1)$-legal condition defined by the function $\max()$.

Let m denote the number of different values that can be proposed. Without loss of generality let $\{1, \ldots, m\}$ be this set of values. Moreover, let n denote the number of processes, and $Comb(n,\beta)$ denote the number of subsets of β elements in a set of n elements (binomial -or Pascal- coefficient).

Let $NB(x,1)$ be the number of input vectors in the $(x,1)$-legal condition defined from $\max()$ (we trivially have $NB(0,1) = m^n$).

Theorem 3 $NB(x,1) = \sum_{\alpha=1}^{m} \sum_{\beta=x+1}^{n} Comb(n,\beta) \times (\alpha-1)^{n-\beta}$.

Proof Let us first determine the value of $NB(1,1)$. As $x=1$, a value has to appear at least twice in a vector in order to be decided. We have the following cases.

- If m is the greatest value in the vector, there are $Comb(n,2) \times (m-1)^{n-2}$ vectors in which the value m appears exactly twice, and more generally, there are $Comb(n,\beta) \times (m-1)^{n-\beta}$ vectors in which m appears exactly β times, for $2 \leq \beta \leq n$.
- If $m-1$ is the greatest value in the vector, there are $Comb(n,\beta) \times (m-2)^{n-\beta}$ vectors in which the value $m-1$ appears exactly β times for $2 \leq \beta \leq n$.
- And similarly, for the cases where $m-2$, $m-3$, ..., $2, 1$ is the greatest value in the vector. (Let us notice that when $\alpha = 1$, i.e., the case of the smallest value, we have a single vector.)
- Summing up all the possible cases, we obtain the following formula[5]:

$$NB(1,1) = \sum_{\alpha=1}^{m} \sum_{\beta=2}^{n} Comb(n,\beta) \times (\alpha-1)^{n-\beta}.$$

  More explicitly, α denotes the greatest value in a vector, β its cardinality in that vector, $Comb(n,\beta)$ the number of vectors with β entries equal to α, and $(\alpha-1)^{n-\beta}$ the number of possibilities for placing the values smaller than α in the vector.

When $x=2$, a simple observation shows that β has to vary from 3 to n, instead of from 2 to n. More generally, a straightforward generalization gives the following formula:

$$NB(x,1) = \sum_{\alpha=1}^{m} \sum_{\beta=x+1}^{n} Comb(n,\beta) \times (\alpha-1)^{n-\beta}.$$

$\square_{Theorem\ 3}$

The computation of $NB(x,\ell)$ (the size of the maximal condition generated by $h_\ell() = \max_\ell()$) is more involved. It is determined in Appendix A.

3 The structure of the sets of (x,ℓ)-legal conditions

This section investigates the structure of the sets of (x,ℓ)-legal conditions, for $0 \leq x < n$ and $0 < \ell \leq n$. The whole picture relating these sets of conditions is described in Figure 1.

Theorem 4 *Let $0 \leq x < n-1$ and $0 < \ell \leq n$. If a condition C is $(x+1,\ell)$-legal, it is also (x,ℓ)-legal.*

Proof As C is $(x+1,\ell)$-legal we have $\forall I \in C : \Sigma_{v \in h_\ell(I)} \#_v(I) > x+1 > x$, from which the (x,ℓ)-density property of C trivially follows. As far as the (x,ℓ)-distance property is concerned, let us observe that the domain of the parameter α used in the statement of the $(x+1,\ell)$-distance property is $\{0,\ldots,x\}$. Hence, the $(x+1,\ell)$-distance property is the addition of the (x,ℓ)-distance property (that addresses the cases $0 \leq \alpha < x$), plus the particular case $\alpha = x$, which completes the proof. $\square_{Theorem\ 4}$

Theorem 5 *Let $0 \leq x < n-1$ and $0 < \ell \leq n$. There are conditions that are (x,ℓ)-legal, but not $(x+1,\ell)$-legal.*

Proof Given a vector I, let $S_\ell(I)$ denote a set of ℓ values appearing in I. Let C be the (x,ℓ)-legal condition, recognized by the function $\max_\ell()$, that contains only the vectors I such that $\forall\, S_\ell(I) :\ \Sigma_{v \in S_\ell(I)} \#_v(I) \leq x+1$.

It is easy to see that C is not empty. Moreover, it follows from the additional constraint on the vectors I that, whatever the function $g_\ell()$, we cannot have $\Sigma_{v \in g_\ell(I)} \#_v(I) > x+1$. So, no $(x+1,\ell)$-recognizing function $g_\ell()$ can be associated with C. It follows that C is not $(x+1,\ell)$-legal. $\square_{Theorem\ 5}$

[5] The case of the single vector that contains only the smallest value (case $\alpha = 1$) appears implicitly in the formula. As $0^{n-\beta} = 0$ for $2 \leq \beta < n$, and $0^{n-\beta} = 1$ for $\beta = n$, when $\alpha = 1$ we have $\sum_{\beta=2}^{n} Comb(n,\beta) \times (\alpha-1)^{n-\beta} = 1$.

Theorems 4 and 5 are associated with the vertical arrow in the figure at the right. In that figure (as in Figure 1), a pair (x, ℓ) represents the set of all the (x, ℓ)-legal conditions, and an arrow from a pair (a, b) to a pair (a', b') means "the set of (a, b)-legal conditions is included in the set of (a', b')-legal conditions". Theorems 6 an 7 (that follow) are associated with the horizontal arrow.

$$\begin{tikzcd} (x+1,\ell) \arrow[r] \arrow[d] & (x+1,\ell+1) \\ (x,\ell) & \end{tikzcd}$$

Theorem 6 *Let $0 \leq x < n$ and $0 < \ell < n$. If a condition C is (x, ℓ)-legal, it is also $(x, \ell + 1)$-legal.*

Proof Let $h_\ell()$ be an (x, ℓ)-recognizing function for the condition C. The proof consists in defining a function $g_{\ell+1}()$ that is $(x, \ell + 1)$-recognizing for C. Let $g_{\ell+1}()$ be defined as follows ($g_{\ell+1}()$ is an appropriate extension of $h_\ell()$). Let $I \in C$.

- If $h_\ell(I) = val(I)$ (i.e., $h_\ell(I)$ contains all the values in I), then $g_{\ell+1}(I) = h_\ell(I)$.
- If $h_\ell(I) \subsetneq val(I)$, then $g_{\ell+1}(I) = h_\ell(I) \cup \{a\}$, where the value a is defined as follows.

  Let I_R be the vector I whose all entries containing a value of $h_\ell(I)$ have been suppressed. (As $h_\ell(I) \subsetneq val(I)$, I_R is a vector with at least one entry). Let $a = f(I_R)$, where f is a deterministic function that extracts a value from a vector.

It directly follows from its definition that $g_{\ell+1}()$ satisfies the $(x, \ell + 1)$-size property for any vector $I \in C$. Let us consider the $(x, \ell + 1)$-density property. Let $I \in C$. We consider two cases.

- $g_{\ell+1}(I) = h_\ell(I)$. As, in that case, we have $h_\ell(I) = val(I)$, we conclude that $\Sigma_{v \in g_{\ell+1}(I)} \#_v(I) = \Sigma_{v \in h_\ell(I)} \#_v(I) = n$. It follows trivially that $\Sigma_{v \in g_{\ell+1}(I)} \#_v(I) > x$.
- $g_{\ell+1}(I) = h_\ell(I) \cup \{a\}$. As $\Sigma_{v \in h_\ell(I)} \#_v(I) > x$, $\Sigma_{v \in g_{\ell+1}(I)} \#_v(I) = \Sigma_{v \in h_\ell(I)} \#_v(I) + \#_a(I)$, and $\#_a(I) \geq 1$, we conclude that $\Sigma_{v \in g_{\ell+1}(I)} \#_v(I) > x + 1 > x$.

To show that $g_{\ell+1}()$ satisfies the $(x, \ell + 1)$-distance property, let us consider a set of vectors $\{I_1, \ldots, I_z\} \subseteq C$ such that $d_G(I_1, \ldots, I_z) = x - \alpha$, with $0 \leq \alpha < x$. We have to show that $\Sigma_{v \in \bigcap_{1 \leq j \leq z} g_{\ell+1}(I_j)} \#_v(\cap_{1 \leq j \leq z} I_j) > \alpha$. According to its definition, we have $g_{\ell+1}(I_j) = h_\ell(I_j) \cup A_j$, with $A_j = \{a_j\}$ or $A_j = \emptyset$. So, $\bigcap_{1 \leq j \leq z} g_{\ell+1}(I_j) = \bigcap_{1 \leq j \leq z} \big(h_\ell(I_j) \cup A_j\big) = \big(\bigcap_{1 \leq j \leq z} h_\ell(I_j)\big) \bigcup \cdots \bigcup \big(\bigcap_{1 \leq j \leq z} A_j\big)$, and consequently $\Sigma_{v \in \bigcap_{1 \leq j \leq z} g_{\ell+1}(I_j)} \#_v(\cap_{1 \leq j \leq z} I_j) \geq \Sigma_{v \in \bigcap_{1 \leq j \leq z} h_\ell(I_j)} l(\cap_{1 \leq j \leq z} I_j)$. Finally, as $d_G(I_1, \ldots, I_z) = x - \alpha \Rightarrow \Sigma_{v \in \bigcap_{1 \leq j \leq z} h_\ell(I_j)} l(\cap_{1 \leq j \leq z} I_j) > \alpha$, we can conclude that $\Sigma_{v \in \bigcap_{1 \leq j \leq z} g_{\ell+1}(I_j)} \#_v(\cap_{1 \leq j \leq z} I_j) > \alpha$, which completes the proof of the $(x, \ell + 1)$-distance property for the function $g_{\ell+1}()$. $\square_{Theorem\ 6}$

Theorem 7 *Let $0 \leq x < n$ and $0 < \ell < n$. There are conditions that $(x, \ell + 1)$-legal, but not (x, ℓ)-legal.*

Proof The proof is similar to the proof of Theorem 5. Given a vector I, let $S_\ell(I)$ denote a set of ℓ values appearing in I. Let C be the $(x, \ell + 1)$-legal condition, recognized by the function $\max_{\ell+1}()$, that contains only the vectors I such that $\forall S_\ell(I) : \ \Sigma_{v \in S_\ell(I)} \#_v(I) \leq x$. It is easy to see that C is not empty. Let us observe that the additional constraint states that no vector of C has a set of ℓ values that appear in it more than x times. It follows from that observation that no function $g_\ell()$ can be (x, ℓ)-recognizing for C (i.e., there is no function $g_\ell()$ such that, for any $I \in C$, $\Sigma_{v \in g_\ell(I)} : \ \#_v(I) > x$). $\square_{Theorem\ 7}$

Theorem 8 *Let $\ell > x$. The set of (x, ℓ)-legal conditions contains the condition including all input vectors.*

Proof Let C_{all} be the condition including all input vectors. Let us consider any vector $I \in C_{all}$. As $\ell > x$, its ℓ greatest values appear at least $\ell \geq x + 1$ times in I. Consequently, when considering the function $h_\ell() = \max_\ell()$, the vectors of C_{all} satisfy the (x, ℓ)-validity and (x, ℓ)-density properties. It then follows from Theorem 2 that C_{all} is (x, ℓ)-legal. $\square_{Theorem\ 8}$

Theorem 9 *The condition made up of all the input vectors is (x, ℓ)-legal only if $\ell > x$.*

Proof Let C_{all} be the condition including all input vectors. We claim that C_{all} cannot be (x, x)-legal. Assuming that claim, we show that C_{all} is not (x, ℓ)-legal for $\ell \leq x$. The proof is by contradiction. Let us assume that C_{all} is (x, ℓ)-legal for $\ell \leq x$. It then follows by successive applications of Theorem 6 that C_{all} is $(x, \ell + 1)$-legal, $(x, \ell + 2)$-legal, ..., (x, x)-legal, contradicting the claim, and proving the theorem.

Proof of the claim. Taking $x = \ell$, Theorem 7 states that there are conditions that are $(x, x + 1)$-legal but not (x, x)-legal. Let C be such a condition. We trivially have $C \subseteq C_{all}$. As C is not (x, x)-legal, it follows that C is not (x, x)-legal either (adding vectors to a condition that is not (x, x)-legal cannot make it (x, x)-legal). $\square_{Theorem\ 9}$

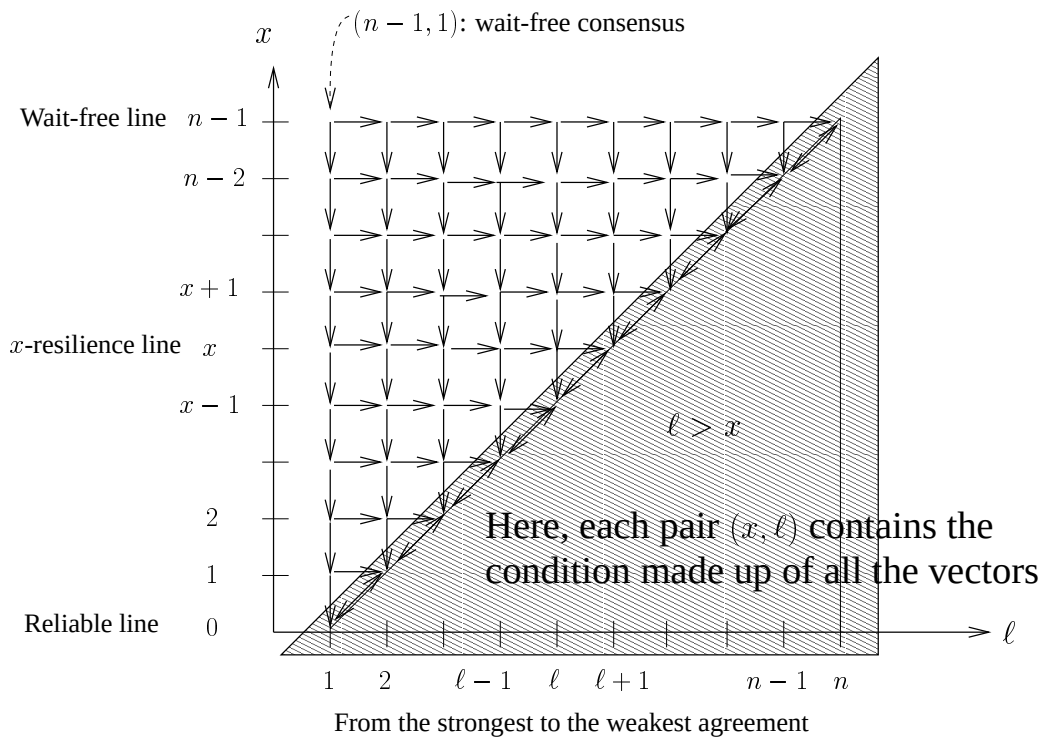

Figure 1: The global inclusion picture, $0 \leq x < n$, $0 < \ell \leq n$

Theorems 4, 5, 6, 7, 8 and 9 are summarized, for all the values of x and ℓ, in the lattice described in Figure 1. It is shown in Appendix B that there are conditions that are (x, ℓ)-legal and not $(x+1, \ell+1)$-legal, and there are conditions that are $(x + 1, \ell + 1)$-legal and not (x, ℓ)-legal.

Theorems 8 and 9 relate the (x, ℓ)-legality of the condition made up of all input vectors with the property $\ell > x$. As already suggested (footnote 3, Introduction), this is the condition-based way to express the impossibility to solve the ℓ-set agreement in an asynchronous system prone to x process crashes when $\ell \leq x$ [5, 13, 28].

4 (x, ℓ)-Legality in asynchronous systems

As indicated in the introduction, when $\ell = 1$, the notion of (x, ℓ)-legality boils down to the notion of x-legality introduced in [20], where it is shown that a condition C allows solving the consensus problem in asynchronous systems prone to x process crashes if and only if C is x-legal.

The asynchronous condition-based algorithm decried in [20] can easily be generalized to solve the ℓ-set agreement problem in asynchronous systems prone to x process crashes, when the input vector belongs to an (x, ℓ)-legal condition. Proving (or disproving), for $\ell > 1$, that (x, ℓ)-legality is necessary for any condition-based solution to the ℓ-set agreement problem remains an open problem.

5 Synchronous hierarchies

A hierarchy on the sets of conditions that allow solving the consensus problem in synchronous systems prone to t process crashes has been established in [22], namely, $\mathcal{S}_t^0 \subset \mathcal{S}_t^1 \subset \cdots \subset \mathcal{S}_t^d \subset \cdots \subset \mathcal{S}_t^t$, where $\mathcal{S}_t^d$ includes all the conditions that allows solving consensus in asynchronous systems prone to $x = t - d$ crashes [20]. (Let us remind that $\mathcal{S}_t^t$ contain the condition that includes all input vectors, and $\mathcal{S}_t^0$ contains all the conditions that allow solving asynchronous consensus despite t process crashes.)

This hierarchy is such that, when the input vector belongs to a condition $C \in \mathcal{S}_t^d$, it is possible to solve optimally synchronous consensus in $d + 1$ rounds. The degree d of a condition is related to its size. There are conditions with more and more vectors when d increases, showing an inherent tradeoff relating the size of a condition and the number of rounds required for the processes to decide.

Considering the (x, ℓ)-legality (instead of x-legality, that corresponds to the case $\ell = 1$), let us define $\mathcal{S}_t^{d,\ell}$ as the set of all $(t - d, \ell)$-legal conditions. Replacing x by $t - d$ in Figure 1, we obtain the following two hierarchies for the ℓ-set agreement problem in synchronous systems prone to up to t process crashes:

- ℓ fixed: $\mathcal{S}_t^{0,\ell} \subset \mathcal{S}_t^{1,\ell} \subset \cdots \subset \mathcal{S}_t^{d,\ell} \subset \cdots \subset \mathcal{S}_t^{t+1-\ell,\ell} = \mathcal{S}_t^{t+2-\ell,\ell} = \cdots = \mathcal{S}_t^{t,\ell}$.
- d fixed: $\mathcal{S}_t^{d,1} \subset \mathcal{S}_t^{d,2} \subset \cdots \subset \mathcal{S}_t^{d,\ell} \subset \cdots \subset \mathcal{S}_t^{d,t-d+1} = \mathcal{S}_t^{d,t-d+2} = \cdots = \mathcal{S}_t^{d,n}$.

6 A synchronous condition-based set agreement algorithm

This section presents a k-set agreement algorithm for synchronous message-passing systems in which up to t processes can crash. The algorithm is instantiated with a condition $C \in \mathcal{S}_t^{d,\ell}$.

6.1 Requirements and properties of the algorithm

Requirements on the values of d and ℓ The values of k and t being fixed, the algorithm considers a condition $C \in \mathcal{S}_t^{d,\ell}$ such that $\ell \leq t - d$. This requirement is motivated by the following observation. On one side, (as announced in the introduction and proved below), the maximal number of rounds required for a process to decide is $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$ when the input vector belongs to the condition. On another side, as $(\ell + d) > t \Rightarrow \left(\lfloor \frac{d-1+\ell}{k} \rfloor + 1 \geq \lfloor \frac{t}{k} \rfloor + 1\right)$, it follows that a synchronous k-set agreement algorithm cannot benefit from a condition such that $\ell + d > t$ ([6]).

The algorithm considers also that $k \geq \ell$. This is because, when $k < \ell$, a condition $C \in \mathcal{S}_t^{d,\ell}$ provides no additional power a k-set algorithm could benefit from in order to expedite decision. (When $k < \ell$, a condition $C \in \mathcal{S}_t^{d,\ell}$ does not restrict enough the set of input vectors in order to obtain a more efficient k-set agreement algorithm.)

Properties of the algorithm Let I be the current input vector, $C \in \mathcal{S}_t^{d,\ell}$ be the condition the algorithm is instantiated with, and f be the number of processes that crash in the current run. As far the number of rounds is concerned, the algorithm guarantees the following properties:

- If $I \in C$:
  - If $f \leq t - d$: no process executes more than two rounds.
  - If $t - d < f$: no process executes more than $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$ rounds.
- $I \notin C$: no process executes more than $\lfloor \frac{t}{k} \rfloor + 1$ rounds. Moreover, if more than $t - d$ processes have crashed before the algorithm starts, no process decides after $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$ rounds.

Remark Let us observe that, up to now, no k-set agreement synchronous algorithm based on consensus conditions $C \in \mathcal{S}_t^d$ has been proposed, that requires at most $\lfloor \frac{d}{k} \rfloor + 1$ rounds. This case (that we have considered in the introduction) appears as a particular instance of the proposed algorithm when considering $\ell = 1$, i.e., when the proposed k-set algorithm is instantiated with a condition that allows solving consensus in an asynchronous system prone to $t - d$ crashes.

[6] This is not at all counter-intuitive. As we have seen in Theorem 8, when $\ell > t - d$ the set $\mathcal{S}_t^{d,\ell}$ contains the condition C_{all} including all the input vectors. As the algorithm depends on the parameters t, ℓ and d only (it does not depend on other parameters of the condition, such as the number of input vectors it is made up), it implicitly assumes that it is instantiated with C_{all}, and the number of rounds is then upper bounded by $\lfloor \frac{t}{k} \rfloor + 1$.

6.2 Synchronous computation model

Round-based synchronous computation The synchronous model is the same as in [22]. The processes $p_1, \ldots, p_n$ communicate and synchronize by sending and receiving messages through channels. Every pair of processes p_i and p_j is connected by a channel. The underlying communication system is assumed to be failure-free: there is no creation, alteration, loss or duplication of message.

The system is *round-based synchronous*, i.e., its executions consists of a sequence of *rounds* identified by the successive integers $1, 2,$ etc. For the processes, the current round number appears as a global variable r that they can read, and whose progress is managed by the underlying system. A round is made up of three consecutive phases:

- A send phase during which each process sends messages.
  We assume that there is a predetermined order in the sending of messages. More precisely each process sends a message first to p_1, then to p_2, etc., until p_n. Thus, if a process crashes during a send phase, only a (possibly empty) prefix of these messages is delivered.
- A receive phase during which each process receives messages.
  The fundamental property of the synchronous model lies in the fact that a message sent by a process p_i to a process p_j at round r, is received by p_j at the same round r.
- A computation phase during which each process processes the messages it received during that round and executes local computation.

A note on the synchronous model of this paper In the standard synchronous model [4, 18, 26], a process p_i sends messages to other processes in each round, and if p_i fails during a round r, any subset of the messages it has sent during r can be lost. In the model used in this paper, each process sends messages to other processes in a predetermined order. This allows the processes to obtain views of the input vector that ordered by containment (this containment is denoted $J1 \leq J2$ in Section 2.1), similarly to what can be obtained when using snapshots in read/write shared memory systems [1]. Actually, only the first round of the algorithm requires this this sending order property. (Let us notice that some lower bound results on synchronous agreement use a similar model where the "adversary" can drop messages in a predetermined order only [8, 16, 19].)

6.3 A synchronous condition-based set agreement algorithm

The synchronous round-based algorithm is described in Figure 2. As indicated, the round number appears as a common variable r whose progress is ensured by the underlying system (lines 2 and 11). The value proposed by the process p_i is denoted v_i.

Given a vector J such that $\#_\perp(J) \leq t$, the predicate $P(J)$ returns $true$, if $\exists I \in C$ such that $J \leq I$. Let us recall that it is possible that several input vectors $I_1, \ldots, I_z \in C$ can exist such that $J \leq I_1, \ldots, J \leq I_z$. In that case, due to Theorem 1 and Definition 4, we have $h_\ell(J) = \bigcap_{1 \leq j \leq z} h_\ell(I_j) \bigcap val(J)$ with $0 < |h_\ell(J)| \leq \ell$.

The algorithm uses the default value $\perp$, that is assumed to be smaller than any value proposed by a process. The notation $a = b \neq \perp$ (resp., $a = b = \perp$) is used as a shortcut for $a = b \wedge a \neq \perp$ (resp., $a = b \wedge a = \perp$).

Local variables and their meaning Each process p_i manages the following local variables.

- $V_i[1 : n]$ is a local array, initialized to $[\perp, \ldots, \perp]$. This array is used only during the first round, namely, if during that round p_i receives the value v_j proposed by p_j, it updates accordingly $V_i[j]$ to v_j (line 5). V_i is the *view* of the input vector obtained by p_i. $\max(V_i)$ denotes the greatest (non-$\perp$) value in V_i.
- v_cond_i, v_tmf_i and v_out_i are three local variables (initialized to $\perp$) whose aim is to contain a proposed value. Their meaning is the following. (In all the cases, p_j is possibly p_i.)
  - When $v\_cond_i = v \neq \perp$, p_i knows that there is a process p_j that, during the first round, obtained a local view V_j such that $\exists I \in C$ with $V_j \leq I$. Consequently, it is possible that the actual input vector belongs to the condition C. So, p_j has computed a value v from $h_\ell(V_j)$ that could be decided (line 6), and that value is currently known by p_i (line 15).
  - When $v\_tmf_i = v \neq \perp$, p_i knows that there is a process p_j that, during the first, round obtained a local view V_j whose number of $\perp$ witnesses $t$oo $m$any $f$ailures (hence the name tmf). That process has consequently computed a value v that could be decided from its view (line 8), and that value is currently known by p_i (line 16).

– When $v\_out_i = v \neq \perp$, p_i knows that there is a process p_j that, during the first, round obtained a local view V_j that allows it to conclude that the input vector does not belong to the condition (line 7). That process p_j has consequently computed a value v that could be decided from its view V_j, and that value is currently known by p_i (line 17).

```
Function Set_Agreement(v_i)

(1)  V_i ← [⊥, ... , ⊥]; V_i[i] ← v_i; v_cond_i ← ⊥; v_out_i ← ⊥; v_tmf_i ← ⊥;
(2)  when r = 1
(3)    begin round
(4)    send (v_i) to p_1, p_2, ... , p_n in that order;
(5)    for_each v_j received do V_i[j] ← v_j end do;
(6)    case (#_⊥(V_i) ≤ t − d) ∧ P(V_i) then v_cond_i ← max(h_ℓ(V_i))
(7)         (#_⊥(V_i) ≤ t − d) ∧ ¬P(V_i) then v_out_i ← max(V_i)
(8)         (#_⊥(V_i) > t − d)           then v_tmf_i ← max(V_i)
(9)    end case
(10) end round;

(11) when r = 2, ... , ⌊t/k⌋ + 1 do
(12) begin round
(13)   send (v_cond_i, v_out_i, v_tmf_i) to p_1, p_2, ... , p_n (in any order);
(14)   if (v_cond_i ≠ ⊥) then return (v_cond_i) end if;
(15)   v_cond_i ← max(v_cond_j received); % ∀v_j : ⊥ < v_j %
(16)   v_tmf_i  ← max(v_tmf_j received);
(17)   v_out_i  ← max(v_out_j received);
(18)   if (r = ⌊(d−1+ℓ)/k⌋ + 1) ∧ v_tmf_i ≠ ⊥ ∧ v_out_i = ⊥) ∨ (r = ⌊t/k⌋ + 1) then
(19)     if (v_cond_i ≠ ⊥) then return (v_cond_i)
(20)     elseif (v_tmf_i ≠ ⊥) then return (v_tmf_i)
(21)                          else return (v_out_i) end if
(22)   end if
(23) end_round
```

Figure 2: A synchronous condition-based set agreement algorithm (code for p_i)

Process state Let UP^r be the set of the processes that have not crashed by the end of the round r, and var_i^r be the value of the local variable var_i of p_i at the end of the round r (if $p_i \in UP^r$). Moreover, for each $p_i \in UP^r$, let $state_i^r$ be the triple $(v\_cond_i, v\_tmf_i, v\_out_i)$. This triple represents the state of p_i from the agreement point of view. Let

$$(state_i^r = state_j^r) \stackrel{def}{=} \left\{ \begin{array}{l} (v\_cond_i^r = v\_cond_j^r \neq \perp) \ \bigvee \\ (v\_cond_i^r = v\_cond_j^r = \perp \ \wedge \ v\_tmf_i{}^r = v\_tmf_j{}^r \neq \perp) \ \bigvee \\ (v\_cond_i^r = v\_cond_j^r = v\_tmf_i{}^r = v\_tmf_j{}^r = \perp \ \wedge \ v\_out_i = v\_out_j \neq \perp) \end{array} \right\}.$$

Underlying principle and process behavior The aim of the algorithm is to ensure that there are no more than k different states when it terminates. To that end, the processes execute consecutive rounds. During each round, they (1) use a classical flood-set technique to disseminate their states, and (2) reduce the number of values in each "class" (the class of v_cond_i variables, the class of v_tmf_i variables, and the class of v_out_i variables) with the help of a deterministic function (namely, $\max()$).

The behavior of a process p_i is fully described in Figure 2. To complete the presentation, let us look at the way a process decides. A process p_i decides when it executes *return* (v) at line 14, 19, 20, or 21.

As suggested in the definition of the equality of two process states, the algorithm establishes a priority on the values defining a process state. If $v\_cond_i \neq \perp$, that value has priority to be decided. If $v\_cond_i = \perp$ and $v\_tmf_j \neq \perp$, v_tmf_j has priority with respect to v_out_i.

When v_cond_i becomes equal to some value $v \neq \perp$, p_i learns that v can be decided from the condition point of view. So, when this occurs, p_i first sent v to all the processes (line 13) and then decides v (14).

In the other cases, a process decides during the round $r = \lfloor \frac{d-1+\ell}{k} \rfloor + 1$ or during the last round $r = \lfloor \frac{t}{k} \rfloor + 1$. In both cases, it decides a value in its state according to the priority mentioned above. In order for p_i to decide during the round $r = \lfloor \frac{d-1+\ell}{k} \rfloor + 1$, some local predicate has to be satisfied (line 18). This predicate states that, to p_i knowledge, (1) there is a process that has seen more than $t - d$ crashes when it executed the first round (this is witnessed by $v\_tmf_i \neq \perp$), and (2) no process can conclude that the input vector is outside the condition (witnessed $v\_out_i = \perp$). This predicate is used to force the decision at round $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$, when, while p_i has not yet decided, there is no evidence that the input vector does not belong to the condition.

Let us finally observe that the algorithm possesses an additional first class property, namely, design simplicity.

7 Proof of the algorithm

In the following, we consider that the algorithm is instantiated with a condition $C \in \mathcal{S}_t^{d,\ell}$. Moreover, we assume $k > \ell$ and $\ell \leq t - d$. The constraint $k > \ell$ is to allow the algorithm to benefit from the condition. The constraint $\ell \leq t - d$ is to eliminate the case where the condition could include all input vectors. As it has been seen in the previous sections, in both cases, the condition-based approach does not allow bypassing the bound $\lfloor \frac{t}{k} \rfloor + 1$.

7.1 Proof of the termination property

Lemma 1 *Let us assume that the input vector belongs to the condition. The protocol described in Figure 2 terminates in (i) 2 rounds when no more than $(t - d)$ processes crash by the end of the first round, and (ii) at most $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$ rounds otherwise.*

Proof

If the input vector belongs to the condition, no process executes line 7, and consequently all local variables v_out_i remain forever equal to $\perp$. Let us partition the set of processes (denoted UP^1) that terminate the first round in two sets: $A = \{p_i \mid v\_cond_i^1 \neq \perp\}$ and $B = \{p_i \mid v\_tmf_i{}^1 \neq \perp\}$. We consider two cases.

- Case $|UP^1| \geq t - d$. As the input vector belongs to the condition, every process that does not crash by the end of the first round belongs to A. Thus, every such process that does not crash proceeds to the second round and decides in line 5, which proves the case (i).
- Case $|UP^1| < n - (t - d)$. As previously, every process in $UP^1 \cap A$ decides in two rounds. Consider now a process $p_j \in B$ that neither crashes nor decides before the round $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$. Such a process p_j set v_tmf_i to a non-$\perp$ value during the first round (line 8) and then that variable remains forever different from $\perp$ (because any process p_k always receives its own value v_tmf_i, line 16). It follows that, when p_j executes the round $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$, the local predicate $v\_tmf_i \neq \perp \wedge v\_out_i = \perp$ is satisfied. The process p_j executes consequently a return() statement (lines 19-21).

$\square_{Lemma\ 1}$

Lemma 2 *Let us assume that the input vector does not belong to the condition. The protocol described in Figure 2 terminates in (i) $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$ rounds when more than $(t - d)$ processes have initially crashed, (ii) and at most $\lfloor \frac{t}{k} \rfloor + 1$ rounds otherwise.*

Proof The fact that the protocol always terminates by round $\lfloor \frac{t}{k} \rfloor + 1$ follows directly from the protocol code (line 18) and the fact that the system is synchronous). So, let us consider the runs where more than $(t - d)$ processes have initially crashed. In that case, each process $p_i \in UP^1$ gets a view V_i^1 with more than $(t - d)$ entries equal to $\perp$, and consequently v_tmf_i is set to a non-$\perp$ value (line 8), while v_cond_i and v_out_i remain forever equal to $\perp$. Hence, for any process p_i that executes the round $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$, we have $v\_tmf_i \neq \perp$ and $v\_out_i = \perp$. It follows that no process executes more than $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$ rounds. $\square_{Lemma\ 2}$

Theorem 10 *The algorithm described in Figure 2 guarantees that no process executes more than $\lfloor \frac{t}{k} \rfloor + 1$ rounds. Moreover, if the input vector belongs to the condition, no process executes more than $\lfloor \frac{d-1+\ell}{k} \rfloor + 1$ rounds, and if additionally no more than $(t - d)$ processes crash, any process executes at most two rounds.*

Proof The proof follows from the Lemmas 1 and 2. $\square_{Theorem\ 10}$

7.2 Proof of the validity property

Theorem 11 *The algorithm described in Figure 2 guarantees that a decided value is a proposed value.*

Proof Let us first observe that, for any process p_i that terminates the first round, one of the local variable v_cond_i, v_tmf_i or v_out_i is different from $\perp$. Moreover, due to the lines 15-17, this property remains satisfied at any round $r \geq 2$.

When they are not equal to $\perp$, the local variables v_tmf_i and v_out_i can contain only a proposed value (this follows from the lines 7, 8, 16, and 17). On another side, when v_cond_i is updated at line 6 during the first round, its new value is a value from the set $h_\ell(V_i)$ that contains only values of V_i (this follows from Theorem 1 and Definition 4).

The validity property follows then directly from the fact that a process decides a non-$\perp$ value from a variable v_cond_i, v_tmf_i or v_out_i. $\square_{Theorem\ 11}$

7.3 Proof of the agreement property

Theorem 12 *The algorithm described in Figure 2 guarantees that at most k values are decided.*

Proof Let a process p_i belong to the set $cond\_winner$ if (1) $v\_cond_i^1 \neq \perp$ and (2) p_i executes line 13 of the second round (i.e., it sends $v\_cond_i^1$ to all the processes). The sets tmf_winner and out_winner are defined similarly; $p_i \in tmf\_winner$ (resp., $p_i \in out\_winner$) if $v\_tmf_i{}^1 \neq \perp$ (resp., $v\_out_i^1 \neq \perp$) and and it executes line 13 of the second round. Let us observe that a process that executes line 13 of the second round belongs to either $cond\_winner$, or tmf_winner, or out_winner). The proof considers three cases.

Case 1: $cond\_winner \neq \emptyset$. Let p_i, p_j, etc., be the processes of the set $cond\_winner$. Due to the sending order during the first round (line 4), the vectors V_i, V_j, ..., are ordered by containment, e.g., $V_i \leq V_j \leq \cdots$. It follows from that containment property and Theorem 1 that $h_\ell(V_i) \supseteq h_\ell(V_j) \supseteq \cdots$. Consequently, as $0 < |h_\ell(V_i)| \leq \ell$ and $\ell \leq k$, no more than k values (1) are deposited in the v_cond_i variables during the first round, and (2) can be decided during the second round.

Let p_m be a process that does not decide during the second round. During that round, p_m receives at least one v_cond_i value that is different from $\perp$. Hence, if it executes the third round, p_m decides one of the (at least one and most ℓ) non-$\perp$ values v_cond_i it has received during the second round, from which it follows that at most $\ell \leq k$ different values can be decided when $cond\_winner \neq \emptyset$.

Case 2: $cond\_winner = \emptyset \wedge out\_winner \neq \emptyset$. We consider two cases.

- A process p_m decides at line 14. Let us observe that this can appear only at a round $r > 2$ (otherwise, we would have $cond\_winner \neq \emptyset$).

  In that case, p_m has sent $v\_cond_m \neq \perp$ to all the processes during r, from which it follows that, from r, only non-$\perp$ v_cond_i values can be decided (they are decided at line 14 or at line 19). It follows from the the previous case discussion (Case 1: $cond\_winner \neq \emptyset$) that there are at most ℓ such $v\_cond_m \neq \perp$ values. The result follows then from $\ell \leq k$.
- No process decides at line 14.

  In that case, the variables v_out_i of all the processes that execute line 17 during the second round are such that $v\_out_i^r \neq \perp$, for $r \geq 2$. Consequently, due to the first predicate of line 18, no process can decide at lines 19-21 before the round $R = \lfloor \frac{t}{k} \rfloor + 1$ round. We claim that, the set $\{state_i^R \mid p_i$ is a process that decides at lines 19-21 during $R\}$ contains at most k different states. It follows from that claim, the definition of the equality of $state_i$ variables, and lines 19-21 that at most k different values can be decided (one from each different $state_i$).

  Proof of the claim. The proof is by contradiction. Let us assume that there are $k+1$ different states at the end of round R. This means that there are k processes that crash during R and there were at least $k+1$ different states at the end of round $R-1$, which in turn means that k processes crashed during $R-1$ and there were at least $k+1$ different states at the end of round $R-1$, etc. until round $r = 1$.

  Let us consider the number of crashes that have to occur during the first round, in order to have $k+1$ different states at the end of that round. In the worst case, as the vectors are ordered by containment, and as each vector

gives rise to a single value (this is due to the use of the function $\max()$ at lines 6-8), $k+1$ different vectors (local views) are needed in order to obtain $k+1$ different states (some vectors providing values $c\_out_i \neq \perp$, others providing values $c\_cond_i \neq \perp$, and others providing values $v\_tmf_i \neq \perp$)[7]. To obtain $k+1$ different vectors, we need to have k crashes during the first round.

When we sum up the number of crashes needed to have $k+1$ different states at the end of round $R = \lfloor \frac{t}{k} \rfloor + 1$, we obtain $k(\lfloor \frac{t}{k} \rfloor + 1)$ process crashes. Let $t = \alpha k + \beta$ with $0 \leq \beta < k$. We have $k \times (\lfloor \frac{t}{k} \rfloor + 1) = k\alpha + k = t - \beta + k > t$, a contradiction with the fact that there are at most t crashes.

Case 3: $cond\_winner = \emptyset \ \wedge \ out\_winner = \emptyset$. As previously, there are two cases. If a process p_m decides at line 14, the proof is the same as the first item of Case 2. So, in the following we consider that no process decides at line 14.

$(cond\_winner = \emptyset \ \wedge \ out\_winner = \emptyset) \Rightarrow (tmf\_winner \neq \emptyset)$, from which follows that any process p_i that terminates the second round is such that $v\_tmf_i^r \neq \perp$ for $r \geq 2$. Consequently (due the priority among the values of the variables v_cond_i, v_tmf_i and v_out_i as defined by the equality on the process states and used in the predicates of lines 19-21), from the second round, the non-$\perp$ values kept in v_out_i (if any) become irrelevant. This means that a process can decide only at line 19 or at line 20.

The rest of the proof consists in showing that at most k values can be decided by these processes. To that end, we show that there are no more than k different states at end of round $R = \lfloor \frac{d-1+\ell}{k} \rfloor + 1$.

Let us assume by contradiction that there are $k+1$ different states (or more) at the end of the round R. This means that there are k crashes during R, and there were at least $k+1$ different states at the end of the round $R-1$. Etc. until the end of the first round.

Let us compute the minimal number of crashes that have to occur during the first round in order to have $k+1$ different states at the end of that round. First, there are at most ℓ distinct values obtained by the processes p_m such that $P(V_m)$ is satisfied. Second, as, from the second round, $v\_tmf_i \neq \perp$ for all the processes p_i, we conclude that there were more than $t-d$ process crashes during the first round. Different vectors V_i such that $(\#_\perp(V_i) > t-d)$ can give rise to different values $v\_tmf_i \neq \perp$.

In order to have $k+1$ different states at the end of $r=1$, we need to have at least $(k+1)-\ell$ different vectors V_i such that $\#_\perp(V_i) > t-d$. As, the vectors are ordered by containment, this means that, in the worst case, we need to have a vector V_{i1} such that $\#_\perp(V_{i1}) = (t-d)+1$, a second vector V_{i2} such that $\#_\perp(V_{i2}) = (t-d)+2$, etc., until a vector V_{ia} such that $\#_\perp(V_{ia}) = (t-d)+(k-\ell+1)$, i.e., at least $(t-d)+(k-\ell+1)$ processes have to crash by the end of the first round.

Let us sum up the number of crashes needed to have at least $k+1$ different states at the end of the round R. There are k crashes from the second round until the round $R = \lfloor \frac{d-1+\ell}{k} \rfloor + 1$, and $(t-d)+(k-\ell+1)$ crashes during the first round. We obtain $S = (t-d+k-\ell+1) + k \times (\lfloor \frac{d-1+\ell}{k} \rfloor)$. Let $d-1+\ell = k\alpha + \beta$ with $0 \leq \beta < k$. $S = (t-d+k-\ell+1) + k\alpha = (t-d+k-\ell+1) + (d-1+\ell-\beta) = t+k-\beta$. A contradiction as $t+k-\beta > t$.

$\square_{Theorem\ 12}$

8 Concluding remarks

Early decision Early decision and early termination address the fact that, while $\lfloor \frac{t}{k} \rfloor + 1$ rounds are necessary in the worst case (when conditions are not used or when the input vector does not belong to the condition), less rounds can be necessary when less than t processes crash. let f, $0 \leq f \leq t$, be the number of processes that crash in the current run. It has been shown that the early deciding lower bound is $R = \min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ [12]. Synchronous early-deciding k-set algorithms can be found in [12, 25, 27].

By using the technique introduced in [22], it is possible to extend the proposed synchronous k-set agreement algorithm in order that, in addition to its previous properties, it never requires more than $\lfloor \frac{f}{k} \rfloor + 2$ rounds.

The main challenge The paper leaves open the following great challenge: show (or disprove) that, when we consider the condition-based approach in the context of asynchronous systems prone to x process crashes, the ℓ-set agreement problem can be solved iff the condition is (x, ℓ)-legal. The "if" part is easy (as already noticed, the consensus algorithm

[7] $k+1$ different vectors can give rise to less than $k+1$ different states, but we are interested in the worst case.

described in [20] designed for x-legal conditions (i.e., the $(x, 1)$-legal in the proposed framework), can easily be generalized to solve the ℓ-set agreement problem when the condition it is instantiated with is (x, ℓ)-legal). The other part is the really difficult side. We spent time and efforts to take up that challenge, but have not yet succeeded.

References

[1] Afek Y., Attiya H., Dolev D., Gafni E., Merritt M. and Shavit N., Atomic Snapshots of Shared Memory. *Journal of the ACM*, 40(4):873-890, 1993.

[2] Aguilera M.K. and Toueg S., A Simple Bivalency Proof that t-Resilient Consensus Requires $t + 1$ Rounds. *Information Processing Letters*, 71:155-178, 1999.

[3] Attiya H. and Avidor Z., Wait-Free n-Set Consensus when Inputs are Restricted. *Proc. 16th Int. Symposium on Distributed Computing (DISC'02)*, Springer Verlag LNCS #2508, pp. 326-338, Toulouse (France), 2002.

[4] Attiya H. and Welch J., *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, McGraw-Hill, 451 pages, 1998.

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

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

[7] Chaudhuri S., Herlihy M., Lynch N. and Tuttle M., Tight Bounds for k-Set Agreement. *Journal of the ACM*, 47(5):912-943, 2000.

[8] Dolev D., Reischuk R. and Strong R., Early Stopping in Byzantine Agreement. *Journal of the ACM*, 37(4):720-741, April 1990.

[9] Fischer M.J. and Lynch N., A Lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters*, 71:183-186, 1982.

[10] Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.

[11] Friedman R., Mostéfaoui A., S. Rajsbaum S. and Raynal M., Asynchronous Agreement and its Relation with Error-Correcting Codes. *IEEE Transactions on Computers*, 56(7):865-876, 2007.

[12] Gafni E., Guerraoui R. and Pochon B., From a Static Impossibility to an Adaptive Lower Bound: The Complexity of Early Deciding Set Agreement. *Proc. 37th ACM Symposium on Theory of Computing (STOC'05)*, Baltimore (MD), May 2005.

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

[14] Izumi T. and Masuzawa T., Condition Adaptation in Synchronous Consensus. *IEEE Transactions on Computers*, 55(7):843-853, 2006.

[15] Izumi T. and Masuzawa T., A Weakly Adaptive Condition-based Consensus Algorithm in Asynchronous Distributed Systems. *Information Processing Letters*, 100(5):199-205, 2006.

[16] Keidar I. and Rajsbaum S., A Simple Proof of the Uniform Consensus Synchronous Lower Bound. *Information Processing Letters*, 85:47-52, 2003.

[17] Loui M.C., and Abu-Amara H.H., Memory Requirements for Agreement Among Unreliable Asynchronous Processes. *Parallel and Distributed Computing: vol. 4 of Advances in Computing Research*, JAI Press, 4:163-183, 1987.

[18] Lynch N.A., Distributed Algorithms. *Morgan Kaufmann Pub.*, San Fransisco (CA), 872 pages, 1996.

[19] Moses, Y. and Rajsbaum, S. A Layered Analysis of Consensus, *SIAM Journal of Computing* 31(4):989-1021, 2002.

[20] Mostefaoui A., Rajsbaum S. and Raynal M., Conditions on Input Vectors for Consensus Solvability in Asynchronous Distributed Systems. *Journal of the ACM*, 50(6):922-954, 2003.

[21] Mostefaoui A., Rajsbaum S. and Raynal M., The Combined Power of Conditions and Failure Detectors to Solve Asynchronous Set Agreement. *Proc. 24th ACM Symposium on Principles of Distributed Computing (PODC'05)*, ACM Press, pp. 179-188, 2005.

[22] Mostefaoui A., Rajsbaum S. and Raynal M., Synchronous Condition-Based Consensus. *Distributed Computing*, 18(5):325-343, 2006.

[23] Mostefaoui A., Rajsbaum S., Raynal M. and Roy M., Condition-Based Protocols for Set Agreement Problems. *Proc. 16th Int. Symposium on Distributed Computing (DISC'02)*, Springer Verlag LNCS #2508, pp. 46-62, Toulouse (France), 2002.

[24] Mostefaoui A., Rajsbaum S., Raynal M. and Roy M., Condition-based Consensus Solvability: a Hierarchy of Conditions and Efficient Protocols. *Distributed Computing*, 17:1-20, 2004.

[25] Raïpin Parvédy Ph., Raynal M. and Travers C., Strongly Terminating Early-Stopping k-set Agreement in Synchronous Systems with General Omission Failures. *Proc. 13th Colloquium on Structural Information and Communication Complexity (SIROCCO'06)*, Springer-Verlag LNCS #4056, pp. 182-196, 2006.

[26] Raynal M., Consensus in Synchronous Systems: a Concise Guided Tour. *Proc. 9th IEEE Pacific Rim Int'l Symposium on Dependable Computing (PRDC'02)*, Tsukuba (Japan), IEEE Computer Press, pp. 221-228, Dec. 2002

[27] Raynal M. and Travers C., Synchronous Set Agreement: a Concise Guided Tour (Including a New Algorithm and a List of Open Problems). *Proc. 12th Int'l IEEE Pacific Rim Dependable Computing Symposium (PRDC'2006)*, IEEE Society Computer Press, pp. 267-274, Riverside (CA), December 2006.

[28] 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.

[29] Zibin Y., Condition-Based Consensus in Synchronous Systems. *Proc. 17th Int. Symposium on Distributed Computing*, Springer-Verlag LNCS #2848, pp. 239-248, Sorrento (Italy), 2003.

A Size of the condition generated by the function $\max_\ell()$

This appendix provides a formula for $NB(x,\ell)$, the size of the (x,ℓ)-legal condition generated by $h_\ell() = \max_\ell()$.

Notation Considering an input vector I, let α_i denote its ith greatest value, and β_i its occurrence number, i.e., $\beta_i = \#_{\alpha_i}(I)$. let us recall that m is the number of different values that can be proposed, and that these values are denoted $1,\ldots,m$. Moreover, let $B_i = \sum_{j=1}^{i} \beta_j$ (the number of occurrences of the i greatest values in a vector; by definition, $B_0 = 0$).

Theorem 13 *$NB(x,\ell) = A + B$ where*

$$A = \sum_{j=1}^{\ell-1} \binom{m}{j} \sum_{\substack{1\le\beta_1\le n-(j-1)\\ 1\le\beta_2\le n-B_1-(j-2)\\ \cdots\\ 1\le\beta_{j-1}\le n-B_{j-2}-1\\ \beta_j=n-B_{j-1}}} \prod_{k=1}^{j} \binom{n-B_{k-1}}{\beta_k}, \quad \textit{and}$$

$$B = \sum_{\substack{\ell\le\alpha_1\le m\\ \ell-1\le\alpha_2\le\alpha_1-1\\ \cdots\\ \ell-(i-1)\le\alpha_i\le\alpha_{i-1}-1\\ \cdots\\ 2\le\alpha_{\ell-1}\le\alpha_{\ell-2}-1\\ 1\le\alpha_\ell\le\alpha_{\ell-1}-1}} \sum_{\substack{1\le\beta_1\le n-(\ell-1)\\ 1\le\beta_2\le n-\beta_1-(\ell-2)\\ \cdots\\ 1\le\beta_i\le n-B_{i-1}-(\ell-i)\\ \cdots\\ 1\le\beta_{\ell-1}\le n-B_{\ell-2}-1\\ \max(1,x+1-B_{\ell-1})\le\beta_\ell\le n-B_{\ell-1}}} (\alpha_\ell-1)^{n-B_\ell} \times \prod_{k=1}^{\ell} \binom{n-B_{k-1}}{\beta_k}.$$

Proof We show that A is the number of vectors that have less than ℓ distinct values (let us notice that all these vectors trivially satisfy the (x,ℓ)-density property), and B is the number of (x,ℓ)-dense vectors generated by the function $\max_\ell()$, that have ℓ or more distinct values.

Determination of A Let A_j, $1 \leq j < \ell$, denote the number of vectors that have exactly j distinct values. We have the following.

- $A_1 = m$. This trivially follows from the fact that there are m different values.
- $A_2 = Comb(m, 2) \sum_{1 \leq \beta_1 \leq n-1} Comb(n, \beta_1)$.

  $Comb(m, 2)$ is the number of sets of two values, taken from the set of m possible values. The greatest of these two values can be placed into one, two, ..., or $n-1$ entries of the vector (hence, $1 \leq \beta_1 \leq n-1$). Moreover, for each value of β_1, there are $Comb(n, \beta_1)$ possibilities to select β_1 entries in the vector of size n. As soon as these β_1 entries have been filled with the greatest value, the entries for the other value are fully determined. Hence, the formula for A_2.
- $A_3 = Comb(m, 3) \sum_{1 \leq \beta_1 \leq n-2} Comb(n, \beta_1) \sum_{1 \leq \beta_2 \leq n-\beta_1-1} Comb(n-\beta_1, \beta_2)$.

  Similarly to the previous case, $Comb(m, 3)$ is the number of distinct sets of three values, and, as we consider the case of a vector containing three distinct values, β_1 varies from 1 to $n-2$ (hence $\sum_{1 \leq \beta_1 \leq n-2}$). There are $Comb(n, \beta_1)$ possibilities to place this value in the vector.

  The number of occurrences β_2 of the second greatest value (out of three) can vary from 1 to $n - \beta_1 - 1$ (hence $\sum_{1 \leq \beta_2 \leq n-\beta_1-1}$). Finally, there are $Comb(n-\beta_1, \beta_2)$ different ways to select entries for that value in the vector. Once the two greatest values (out of three) have been placed in a vector, the entries for the third value are fully determined.
- Similarly we have:

$$\begin{aligned} A_4 \quad = \quad & Comb(m, 4) \sum_{1 \leq \beta_1 \leq n-3} Comb(n, \beta_1) \sum_{1 \leq \beta_2 \leq n-\beta_1-2} Comb(n-\beta_1, \beta_2) \times \\ & \sum_{1 \leq \beta_3 \leq n-\beta_1-\beta_2-1} Comb(n-\beta_1-\beta_2, \beta_3). \end{aligned}$$

- And so on until

$$\begin{aligned} A_{\ell-2} \quad = \quad & Comb(m, \ell-2) \times \\ & \sum_{1 \leq \beta_1 \leq n-(\ell-3)} Comb(n, \beta_1) \times \\ & \quad \sum_{1 \leq \beta_2 \leq n-\beta_1-(\ell-4)} Comb(n-\beta_1, \beta_2) \times \\ & \quad\quad \sum_{1 \leq \beta_3 \leq n-\beta_1-\beta_2-(\ell-5)} Comb(n-\beta_1-\beta_2, \beta_3) \times \\ & \quad\quad\quad \cdots \times \\ & \quad\quad\quad\quad \sum_{1 \leq \beta_{\ell-3} \leq n-\beta_1-\beta_2-\cdots-\beta_{\ell-4}-1} Comb(n-\beta_1-\cdots-\beta_{\ell-4}, \beta_{\ell-3}). \end{aligned}$$

  As in the previous cases, as soon as $\ell - 3$ values have been placed in a vector, the entries for the last value (the smallest value in the set of $(\ell-2)$ values to be placed) are fully determined.
- And finally

$$\begin{aligned} A_{\ell-1} \quad = \quad & Comb(m, \ell-1) \times \\ & \sum_{1 \leq \beta_1 \leq n-(\ell-2)} Comb(n, \beta_1) \times \\ & \quad \sum_{1 \leq \beta_2 \leq n-\beta_1-(\ell-3)} Comb(n-\beta_1, \beta_2) \times \\ & \quad\quad \cdots \times \\ & \quad\quad\quad \sum_{1 \leq \beta_{\ell-2} \leq n-\beta_1-\beta_2-\cdots-\beta_{\ell-3}-1} Comb(n-\beta_1-\cdots-\beta_{\ell-3}, \beta_{\ell-2}). \end{aligned}$$

Summing up $A_1 + \cdots + A_{\ell-1}$, using classical commutativity and associativity rules, and factoring terms, we obtain the formula A stated in the theorem[8].

Determination of B Let us now consider the number of vectors, generated by the function $\max \ell()$, that have ℓ or more distinct values. As before, let α_i denote the ith greatest value in the vector. As the vectors we consider now have at least ℓ distinct values, we have the following possibilities for the ℓ greatest values values $\alpha_1, \ldots, \alpha_\ell$ and their occurrence numbers $\beta_1, \ldots, \beta_\ell$:

[8] In the case of the vectors that contain exactly $\ell - 1$ distinct values, the formula A uses the fact that $\beta_1 + \cdots + \beta_{\ell-2} + \beta_{\ell-1} = n$, i.e., $\beta_{\ell-1} = n - (\beta_1 + \cdots + \beta_{\ell-2})$, and then $Comb(n - \beta_1 - \cdots - \beta_{\ell-2}, \beta_{\ell-1}) = Comb(\beta_{\ell-1}, \beta_{\ell-1}) = 1$. This appears in the formula A in the case $j = \ell - 1$.

Input vectors of C	$h_1()$
$I_1 = [a, a, c, d]$	$h_1(I_1) = \{a\}$
$I_2 = [b, b, c, d]$	$h_1(I_2) = \{b\}$
$I_3 = [a, b, c, c]$	$h_1(I_3) = \{c\}$
$I_4 = [a, b, d, d]$	$h_1(I_4) = \{d\}$

Table 1: A $(1, 1)$-legal condition C

i	possible values for α_i	corresponding occurrence number β_i
1	$\ell \leq \alpha_1 \leq m$	$1 \leq \beta_1 \leq n - (\ell - 1)$
2	$\ell - 1 \leq \alpha_2 \leq \alpha_1 - 1$	$1 \leq \beta_2 \leq n - \beta_1 - (\ell - 2)$
...	...	...
$\ell - 1$	$2 \leq \alpha_{\ell-1} \leq \alpha_{\ell-2} - 1$	$1 \leq \beta_{\ell-1} \leq n - \beta_1 - \beta_2 - \cdots - \beta_{\ell-2} - 1$
ℓ	$1 \leq \alpha_\ell \leq \alpha_{\ell-1} - 1$	$\max(1, x + 1 - \beta_1 - \cdots - \beta_{\ell-1}) \leq \beta_\ell \leq n - \beta_1 - \beta_2 - \cdots - \beta_{\ell-1}$

Let us observe that $\alpha_1 \geq \ell$ is due to the fact that the vectors considered here contain at least ℓ different values. Similarly, $\alpha_2 \geq \ell - 1$ is due to the fact that the vectors whose greatest value is ℓ contain $\ell - 1$ values different from ℓ. Etc. for α_3 until α_ℓ.

The value of β_1 has not to bypass $n - (\ell - 1)$ in order to leave available enough entries of the vector for the other values (there are at least $\ell - 1$ such values). The same observation applies to each other value β_i, $i \leq \ell$ (β_2 has not to bypass $n - \beta_1 - (\ell - 2)$, etc.)

On another side, α_ℓ (the smallest of the ℓ greatest values) has to be present enough in order these ℓ values appear at least $x + 1$ times in a vector (this is required by the (x, ℓ)-density property). So, when $x + 1 > (\beta_1 + \cdots + \beta_{\ell-1})$, we need to have $\beta_\ell \geq x+1-(\beta_1+\cdots+\beta_{\ell-1})$. Hence, the lowest value that β_ℓ can have is $\max(1, x+1-\beta_1-\cdots-\beta_{\ell-1})$.

According to these observations on the values of α_i and β_i, $1 \leq i \leq \ell$, the number of vectors with at least ℓ distinct values is equal to the following "sigma of sigmas" quantity:

$$\begin{array}{l}
\sum_{\ell \leq \alpha_1 \leq m} \sum_{1 \leq \beta_1 \leq n-(\ell-1)} Comb(n, \beta_1) \times \\
\sum_{\ell-1 \leq \alpha_2 \leq \alpha_1 - 1} \sum_{1 \leq \beta_2 \leq n-\beta_1-(\ell-2)} Comb(n - \beta_1, \beta_2) \times \\
\sum_{\ell-2 \leq \alpha_3 \leq \alpha_2 - 1} \sum_{1 \leq \beta_3 \leq n-\beta_1-\beta_2-(\ell-3)} Comb(n - \beta_1 - \beta_2, \beta_3) \times \\
\qquad \cdots \times \\
\sum_{1 \leq \alpha_{\ell-1} \leq \alpha_{\ell-2}-1} \sum_{1 \leq \beta_{\ell-1} \leq n-\beta_1-\cdots-\beta_{\ell-2}-1} Comb(n - \beta_1 - \cdots - \beta_{\ell-2}, \beta_{\ell-1}) \times \\
\sum_{1 \leq \alpha_\ell \leq \alpha_{\ell-1}-1} \sum_{\max(1, x+1-\beta_1-\cdots-\beta_{\ell-1}) \leq \beta_\ell \leq n-\beta_1-\beta_2-\cdots-\beta_{\ell-1}} Comb(n - \beta_1 - \cdots - \beta_{\ell-1}, \beta_\ell) \times \\
(\alpha_\ell - 1)^{n-\beta_1-\cdots-\beta_\ell}.
\end{array}$$

The ℓ greatest values used in a vector are $\alpha_1, \ldots, \alpha_\ell$. So, given such a set of ℓ values, the last term (namely, $(\alpha_\ell - 1)^{n-\beta_1-\cdots-\beta_\ell}$) represents the number of different vectors that are due to the $\alpha_\ell - 1$ remaining values (these values are smaller than or equal to $\alpha_\ell - 1$). After rewriting this sum, we obtain the quantity denoted B in the statement of the theorem. $\square_{Theorem\ 13}$

Given a pair (x, ℓ), the value of $NB(x, \ell)$ can easily be computed using a mathematical software.

B (x, ℓ)-Legality vs $(x + 1, \ell + 1)$-legality

Theorem 14 *There are conditions that are (x, ℓ)-legal and not $(x + 1, \ell + 1)$-legal.*

Proof The proof consists in exhibiting a counter-example. Let us consider a system made up of $n = 4$ processes, and $x = \ell = 1$.

Let C be the condition composed of the four vectors described in Table 1 and $h_1()$ the function defined in the second column of that table. It is easy to see that this function $h_1()$ is a recognizing function for C for the pair of parameters $x = \ell = 1$, and consequently C is $(1, 1)$-legal.

We show that C is not $(2, 2)$-legal. To that end let us try to define an appropriate function $h_2()$.

- Definition of $h_2(I_1)$. As $h_2(I_1)$ has to include no more than two elements, we have $a \in h_2(I_1)$ (otherwise the density property could not be satisfied). For the second element of $h_2(I_1)$, we have the choice between c or d. We take arbitrarily $h_2(I_1) = \{a, c\}$. (Due to the symmetry relating I_1 and I_2 on one side, and I_3 and I_4 on the other side, if we had chosen $h_2(I_1) = \{a, d\}$, we should exchange the vectors I_3 and I_4 in the reasoning that follows.)
- Definition of $h_2(I_2)$. Trivially, $h_2(I_2)$ has to include b. To select the second element of $h_2(I_2)$, we have to take into account the fact $d_G(I_1, I_2) = 2$, in order the $(2, 2)$-distance property be satisfied. As $d_G(I_1, I_2) = x - \alpha = 2 \Rightarrow \alpha = 0$, we must have $\Sigma_{v \in h_\ell(I_1) \cap h_\ell(I_2)} \#_v(I_1 \cap I_2) > 0$. It follows that c has to be selected, and we have consequently $h_2(I_2) = \{b, c\}$.
- Definition of $h_2(I_4)$. Similarly to the previous cases, $h_2(I_4)$ has to include d.

  Moreover, as $d_G(I_1, I_4) = x - \alpha = 2$ (i.e., $\alpha = 0$), we must have $\Sigma_{v \in h_\ell(I_1) \cap h_\ell(I_4)} \#_v(I_1 \cap I_4) > 0$. It follows that $h_2(I_4) = \{a, d\}$.

  But as $d_G(I_2, I_4) = x - \alpha = 2$ (i.e., $\alpha = 0$), we must also have $\Sigma_{v \in h_\ell(I_2) \cap h_\ell(I_4)} \#_v(I_2 \cap I_4) > 0$, with means that $h_2(I_4) = \{d, c\}$.

  So, $h_2(I_4)$ has to equal to both $\{a, d\}$ and $\{d, c\}$: an impossibility.

It follows that no recognizing function can be associated with C for $x = \ell = 2$. The condition C is not $(2, 2)$-legal.

$\square_{Theorem\ 14}$

Theorem 15 *Let $0 \leq x < n-1$ and $0 < \ell \leq x$. There are conditions that are $(x+1, \ell+1)$-legal and not (x, ℓ)-legal.*

Proof As x and ℓ are such that $0 < \ell \leq x < n - 1$, we have $\ell + 1 \leq n - x + \ell - 1$. Let $v_1, v_2, \ldots, v_{n-x+\ell-1}$ be a set of $n - x + \ell - 1$ different values. Let us consider the condition C made up of the $\ell + 1$ following vectors with n entries. All these vectors differ in their $x - \ell + 1$ first entries only. More precisely, they are such that:

- Different part: $1 \leq j \leq x - \ell + 1 \;\Rightarrow\; I_1[j] = v_1 \;\wedge\; I_2[j] = v_2 \;\wedge\; \cdots \;\wedge\; I_{\ell+1}[j] = v_{\ell+1}$.
- Common part: $1 \leq j \leq n - x + \ell - 1 \;\Rightarrow\; I_1[x - \ell + 1 + j] = \cdots I_{\ell+1}[x - \ell + 1 + j] = v_j$.

It is easy to verify that C is $(x+1, \ell+1)$-recognized by the function $h_{\ell+1}()$ defined as follows: $\forall\, j :\; h_{\ell+1}(I_j) = \{v_1, v_2, ..., v_{\ell+1}\}$.

We now show that C is not (x, ℓ)-legal. The proof is by contradiction. Let us assume that there is an (x, ℓ)-recognizing function $g_\ell()$ for C. For each vector $I_j \in C$ $(1 \leq j \leq \ell + 1)$ we have $v_j \in g_\ell(I_j)$ (this follows from the fact that $\ell \leq x$ and v_j is the only value that appears more than once in I_j).

When we apply the (x, ℓ)-distance property to the whole set of vectors defining the condition, we have $d_G(I_1, \ldots, I_{\ell+1}) = x - \ell + 1 = x - \alpha$. Consequently, the (x, ℓ)-recognizing function $g_\ell()$ is such that $\Sigma_{v \in \bigcap_{1 \leq j \leq \ell+1} g_\ell(I_j)} \#_v(\cap_{1 \leq j \leq \ell+1} I_j) > \alpha = \ell - 1$ (Observation O). We have also the following.

- Due to (x, ℓ)-size property, each $g_\ell(I_j)$ has at most ℓ values. Moroever, in order to have $|\bigcap_{1 \leq j \leq \ell+1} g_\ell(I_j)| = \ell$, we need to have $g_\ell(I_1) = g_\ell(I_2) = \cdots = g_\ell(I_{\ell+1})$. But, these sets are not all equal because $\bigcup_{1 \leq j \leq \ell+1} g_\ell(I_j)$ contains $\ell + 1$ different values. It follows that the set $\bigcap_{1 \leq j \leq \ell+1} g_\ell(I_j)$ contains at most $\ell - 1$ values.
- On another side, as (from the definition of the vectors) $\bigcap_{1 \leq j \leq \ell+1} I_j = [v_1, v_2, \cdots, v_{n-x+\ell-1}]$, that intersecting vector contains only distinct values.

It follows that $\Sigma_{v \in \bigcap_{1 \leq j \leq \ell+1} h_\ell(I_j)} \#_v(\cap_{1 \leq j \leq \ell+1} I_j) \leq \ell - 1$, which contradicts Observation O, and completes the proof.

$\square_{Theorem\ 15}$