Minimizing the Number of Opinions for Fault-Tolerant Distributed Decision Using Well-Quasi Orderings

Pierre Fraigniaud, Sergio Rajsbaum, Corentin Travers

Minimizing the Number of Opinions for Fault-Tolerant Distributed Decision Using Well-Quasi Orderings[*]

Pierre Fraigniaud
Univ. Paris Diderot and CRNS, France
pierre.fraigniaud@liafa.univ-paris-diderot.fr

Sergio Rajsbaum
Instituto de Matemáticas, UNAM, Mexico
rajsbaum@im.unam.mx

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

Abstract

The notion of deciding a *distributed language* $\mathcal{L}$ is of growing interest in various distributed computing settings. Each process p_i is given an input value x_i, and the processes should collectively decide whether their set of input values $x = (x_i)_i$ is a valid state of the system w.r.t. to some specification, i.e., if $x \in \mathcal{L}$. In *non-deterministic* distributed decision each process p_i gets a local certificate c_i in addition to its input x_i. If the input $x \in \mathcal{L}$ then there exists a certificate $c = (c_i)_i$ such that the processes collectively accept x, and if $x \notin \mathcal{L}$, then for every c, the processes should collectively reject x. The collective decision is expressed by the set of *opinions* emitted by the processes, and one aims at minimizing the number of possible opinions emitted by each process.

In this paper we study non-deterministic distributed decision in asynchronous systems where processes may crash. In this setting, it is known that the number of opinions needed to deterministically decide a language can grow with n, the number of processes in the system. We prove that every distributed language $\mathcal{L}$ can be non-deterministically decided using only three opinions, with certificates of size $O(\log \alpha(n))$ bits, where α grows at least as slowly as the inverse of the Ackerman function. The result is optimal, as we show that there are distributed languages that cannot be decided using just two opinions, even with arbitrarily large certificates.

To prove our upper bound, we introduce the notion of *distributed encoding of the integers*, that provides an explicit construction of a long *bad sequence* in the *well-quasi-ordering* $(\{0,1\}^*, =)$ controlled by the successor function. Thus, we provide a new class of applications for well-quasi-orderings that lies outside logic and complexity theory. For the lower bound we use combinatorial topology techniques.

Keywords: distributed decision, distributed verification, well-quasi-ordering.

[*] Supported by ECOS-CONACYT Nord grant M12A01, ANR project DISPLEXITY, INRIA project GANG and UNAM-PAPIIT grant.

1 Introduction

Fault-tolerant distributed decision In *distributed decision* each process has only a local perspective of the system, and collectively the processes have to decide if some predicate about the global system state is valid. Recent work in this area includes but is not limited to, deciding locally whether the nodes of a network are properly colored , checking the results obtained from the execution of a distributed program [13, 15], designing time lower bounds on the hardness of distributed approximation [7], estimating the complexity of logics required for distributed run-time verification [14], and elaborating a distributed computing complexity theory [12, 16].

More specifically, the predicate to be decided is specified as a *distributed language* $\mathcal{L}$. Each process p_i is given an input value x_i, and the processes should collectively decide whether their set of input values $x = (x_i)_i$ represents a valid state of the system w.r.t. to the specification, i.e., if $x \in \mathcal{L}$. The collective decision is expressed by the set of *opinions* emitted by the processes. Often processes emit two possible opinions, TRUE or FALSE, and the collective opinion is interpeted as the conjunction of the emitted values. Indeed, in a distributed system where processes are unable to agree on what the global system state is (e.g. due to failures, communication delays, locality, etc), it is unavoidable that processes have different opinions about the validity of the predicate at any given moment (a consequence of consensus impossibility [10]). Some languages $\mathcal{L}$ may be decided by emitting only two opinions, but not always (e.g., in the context of linear temporal logic, [3]). In fact, it is known that up to n different opinions may be necessary to decide some languages [14]. For example, for the *k-set agreement* language, specifying that at most k leaders are elected, $\min\{2k, n\} + 1$ opinions are necessary and sufficient in a system where n asynchronous processes may crash [15]. In general, one aims at minimizing the number of possible opinions.

Non-deterministic distributed decision In *non-deterministic* distributed decision, each process p_i gets a local certificate c_i in addition to its input x_i. If the input $x \in \mathcal{L}$ then there exists a certificate $c = (c_i)_i$ such that the processes collectively accept x, and if $x \notin \mathcal{L}$, then for every c, the processes should collectively reject x (i.e., the protocol cannot be fooled by "fake" certificates on illegal instances). Notice that as for the input, the certificate is also distributed; each process only knows its local part of the certificate. As in the deterministic case, the collective decision is expressed by the set of opinions emitted by the processes.

This non-determinism framework is inspired by classical complexity theory, but it has been used before also in various distributed settings, e.g. in distributed complexity [12], in silent self-stabilization [4] (as captured by the concept of proof-labeling schemes [20]), as well as failure detectors [5] where an underlying layer is charged with producing certificates giving information about process failures — the failure detector should provide certificates giving sufficient information about process failures to solve e.g. consensus, but an incorrect certificate should not lead to an invalid consensus solution.

In several of these contexts, one seeks certificates that are as small as possible, perhaps for information theoretic purposes, privacy purposes, or because certificates have to be exchanged among processes [4, 20]. As we shall prove in this paper, it is possible to use small certificates to enable the number of opinions to be drastically reduced. We do so in the standard framework of asynchronous crash-prone processes communicating by writing and reading shared variables[1].

[1] The theory of read/write wait-free computation is of considerable significance, because results in this model can be transferred to other message-passing and f-resilient models e.g. [2, 17].

Our Contribution We show that, for every distributed language $\mathcal{L}$, it is possible to design a non-deterministic protocol using certificates of few bits, while using a small set of opinions. Our solution is based on a combinatorial construction called a *distributed encoding.*

We define a distributed encoding of the integers as a collection of code-words providing every integer n with a code $w = (w_i)_{i=1,\dots,n}$ in Σ^n, where Σ is a (possibly infinite) alphabet, such that, for any $k \in [1, n)$, no subwords[2] $w' \in \Sigma^k$ of w is encoding k. Trivially, every integer $n \geq 1$ can be (distributedly) encoded by the word $w = (\text{bin}(n), \dots, \text{bin}(n)) \in \Sigma^n$ with $\Sigma = \{0,1\}^*$, where $\text{bin}(n)$ is the binary representation of n. Hence, to encode the first n integers, one can use words on an alphabet with n symbols, encoded on $O(\log n)$ bits. Our first result is a constructive proof that there is a distributed encoding of the integers which encodes the first n integers using words on an alphabet with symbols on $O(\log \alpha(n))$ bits, where α is a function growing at least as slowly as the inverse-Ackerman function. This first result is obtained by considering the *well-quasi-ordering* $(\Lambda, =)$ where $\Lambda = \{0, 1\}$ is composed of two incomparable elements 0 and 1, and by constructing long (so-called) *bad sequences* of words over $(\Lambda^*, \leq_*)$ starting from any words $a \in \Lambda^*$, and controlled by the successor function $g(x) = x + 1$. (See Section 2 for the formal definitions of these concepts, and for the definition of the relation $\leq_*$ over Λ^*).

Our main contribution is an application of distributed encoding of the integers to distributed computing. This provides in turn a new class of applications for well-quasi-orderings, that lies outside logic and complexity theory. Specifically, we prove that any distributed language $\mathcal{L}$ can be non-deterministically decided with certificates on $O(\log \alpha(n))$ bits, and only three opinions. Each opinion provides an estimation of the correctness of the execution from the perspective of one process. Moreover, using argument from combinatorial topology, we show that the result is best possible. Namely, there are distributed languages for which two opinions are insufficient, even with at most three processes, and regardless to the size of the certificates.

To sum up, using well-quasi-orderings, and the novel notion of distributed encoding of the integers, we prove that non-deterministic distributed decision is possible for any language, with certificates of practically constant size, by letting each process choose one opinion among just three, which is optimal.

This motivates a new line of research in distributed computing, consisting in designing distributed algorithms producing *certified* outputs, i.e., outputs that can be verified afterward by another algorithm. This can be achieved in the framework of asynchronous systems with *transient* failures [4]. Our results demonstrate that, conceptually, this can also be achieved in asynchronous systems with *crash* failures, at low costs, in term of both certificate size and number of opinions.

Related work Deterministic distributed decision in the context of asynchronous, crash-prone distributed computing was introduced in [13] with the name *checking*, where a characterization of the tasks that are AND-checkable is provided. The results where later on extended in [15] to the set agreement task and in [14] proving nearly tight bounds on the number of opinions required to check any distributed language. In [12, 16] the context of *local* distributed network computing [23] is considered. It was shown that not all network decision tasks can be solved locally by a non-deterministic algorithm. On the other hand, every languages can be locally decided non-deterministically if one allows the verifier to err with some probability. See [11] for more on randomized local distributed decision, and [8] for its impact on construction tasks.

[2] Such a subword is of the form $w' = (w_{i_j})_{j=1,\dots,k}$ with $i_j < i_{j+1}$ for $j \in [1, k)$.

Our construction of distributed encoding of the integers relies very much on the notion of *well-quasi-ordering* (wqo) [21].This important tool for logic and computability has a wide variety of applications — see [22] for a survey. One important application is providing *termination arguments* in decidability results [6]. Indeed, thirteen years after publishing his undecidability result, Turing [28] proposed the now classic method of proving program termination using so-called *bad sequences*, with respect to a wqo. In this setting, the problem of *bounding* the length of bad sequences is of utmost interest as it yields upper bounds on terminating program executions. Hence, the interest in *algorithmic aspects* of wqos has grown recently [9, 24], as witnessed by the amount of work collected in [25]. Our paper is tackling the study of wqos, from a *distributed algorithm* perspective. Also, lower bounds showing Ackermanian termination growth have been identified in several applications, including lossy counter machines and reset Petri nets [25, 27].

2 Distributed Encoding of the Integers

In this section, we define the notion of distributed encoding of the integers, and construct a *compact* distributed encoding.

Given a finite or infinite alphabet Σ, a *word* of size n on Σ is an ordered sequence $w = w_1, w_2, \dots, w_n$ of symbols $w_i \in \Sigma$. The set of all finite words over Σ is Σ^*, and the set of all words of size n is Σ^n. A *sub-word of* w is a word $w' \in \Sigma^*$, which is sub-sequence of w, $w' = w_{i_1}, w_{i_2}, \dots, w_{i_k}$ with $k < n$ and $1 \leq i_1 < i_2 < \cdots < i_k \leq n$.

Definition 2.1. *A* distributed encoding *of the positive integers is a pair* (Σ, f) *where* Σ *is a (possibly infinite) alphabet, and* $f : \Sigma^* \to \{\text{true}, \text{false}\}$ *satisfying that, for every integer* $n \geq 1$*, there exists a word* $w \in \Sigma^n$ *with* $f(w) = \text{true}$*, such that for every sub-word* w' *of* w*,* $f(w') = \text{false}$*. The word* w *is called the* distributed code *of* n*.*

A trivial distributed encoding of the integers can be obtained using the infinite alphabet $\Sigma = \{0,1\}^*$ (each symbol is a sequence of 0's and 1's). The distributed code of n consists in repeating n times the binary encoding of n, for each positive integer n, $w = \text{bin}(n), \dots, \text{bin}(n)$. For every integer $n \geq 1$ and every word $w \in \Sigma^n$, we set $f(w) = \text{true}$ if and only if $w_i = \text{bin}(n)$ for every $i \in \{1, \dots, n\}$. However, this encoding is quite redundant, and consumes an alphabet of n symbols to encode the first n positive integers (i.e., $O(\log n)$ bits per symbol).

A far more compact distributed encoding of the integers can be obtained, using a variant of the Ackermann function. Given a function $f : \mathbb{N} \to \mathbb{N}$, we denote by $f^{(n)}$ the nth iterate of f, with $f^{(0)}$ the identity function. Let $A_k : \mathbb{N} \to \mathbb{N}, k \geq 1$ be the family of functions defined recursively as follows:

$$A_k(n) = \begin{cases} 2n+2 & \text{if } k = 1 \\ A_{k-1}(\dots A_{k-1}(0)) = A_{k-1}^{(n+1)}(0) & \text{otherwise.} \end{cases} \tag{1}$$

Hence $A_k(0) = 2$ for every $k \geq 1$, and, for $n \geq 0$, $A_2(n) = 2^{n+2} - 2$, and $A_3(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} - 2$, where the a tower is of height $n+2$. (Many versions of the Ackerman function exist, and a possible definition [26] is $Ack(n) = A_n(1)$). Let $F : \mathbb{N} \to \mathbb{N}$ be the function: $F(k) = A_1(A_2(\dots(A_{k-1}(A_k(0))))) + 1$. Finally, let $\alpha : \mathbb{N} \to \mathbb{N}$ be the function:

$$\alpha(k) = \min\{i \geq 1 : F^{(i)}(1) > k\}. \tag{2}$$

Hence, α grows extremely slowly, and, for any reasonable value of n, we have $\alpha(n) \leq 5$. In addition, note that $F^{(n)}(1) > n$ for every $n \geq 1$. Hence, a crude lower bound of $F^{(n)}(1)$ is $F^{(n)}(1) \geq Ack(n-1)$. Therefore the function α grows at least as slowly as the inverse-Ackerman function.

Theorem 2.2. *There is a distributed encoding (Σ, f) of the positive integers which encodes the first n integers using words on an alphabet with symbols on $O(\log \alpha(n))$ bits, where α is defined in Eq. (2).*

The proof of Theorem 2.2 heavily relies on the notion of *well-quasi-ordering.* Recall that a *well-quasi-ordering* (wqo) is a quasi-ordering that is well-founded. That is, a wqo is a pair $(A, \leq)$, where $\leq$ is a reflexive and transitive relation over a set A, such that every infinite sequence of elements $a^{(0)}, a^{(1)}, a^{(2)}, \cdots$ from A contains an *increasing pair*, i.e., a pair $(a^{(i)}, a^{(j)})$ with $i < j$ and $a^{(i)} \leq a^{(j)}$. Sequences (finite or infinite) with an increasing pair of elements are called *good* sequences. Instead, sequences where no such increasing pair can be found are called *bad.* Therefore, every infinite sequence over a wqo A is good, and, as a consequence, bad sequences over a wqo A are finite. Often, $a \in A$ is a finite word over some domain Λ, i.e., $a \in \Lambda^*$. Assuming $(\Lambda, \leq)$ itself is a wqo, then Higman's lemma says that $(\Lambda^*, \leq_*)$ is a wqo, where $\leq_*$ is the *subword* ordering defined as follows. For any $a = a_1, a_2, \ldots, a_n \in \Lambda^*$, and any $b = b_1, b_2, \ldots, b_m \in \Lambda^*$,

$$a \leq_* b \iff \exists\, 1 \leq i_1 < i_2 < \cdots < i_n \leq m : (a_1 \leq b_{i_1}) \wedge \cdots \wedge (a_n \leq b_{i_n}).$$

As said before, the longest bad sequence starting on any $a \in \Lambda^*$ is of interest for practical applications (e.g., to obtain upper bounds on the termination time of a program). This length is strongly related to the *growth* of the words' length in Λ^*. More generally, let $|\cdot|$ be a norm on a wqo A that defines the *size* $|a|$ of each $a \in A$. For any $a \in A$, there is a longest bad sequence $a^{(0)}, a^{(1)}, a^{(2)}, \ldots, a^{(k)}$ starting on $a^{(0)} = a$, provided that, for every $i \geq 0$, the size of $a^{(i+1)}$ does not grow unboundedly with respect to the size of the previous element $a^{(i)}$. Given an increasing function g, the *length function* $L_g(n)$ is defined as the length of the longest sequence over all sequences *controlled* by g, starting in an element a with $|a| \leq n$. The function g controls the sequence in the sense that it bounds the growth of elements as we iterate through the sequence. That is, $L_g(n)$ is the length of the longest sequence $a^{(0)}, a^{(1)}, \ldots$ such that $|a^{(0)}| \leq n$, and, for any $i \geq 0$, $|a^{(i+1)}| \leq g(|a^{(i)}|)$. The Length Function Theorem of [24] provides an upper bound on bad sequences parametrized by a control function g and by the size $p = |\Lambda|$ of the alphabet.

Theorem 2.2 is obtained by Considering the *well-quasi-ordering* $(\Lambda, =)$ where $\Lambda = \{0, 1\}$ is composed of two incomparable elements 0 and 1. We construct a bad sequence $B(a)$ of words over $(\Lambda^*, \leq_*)$ starting from any words $a \in \Lambda^*$, and controlled by the successor function $g(x) = x + 1$. That is, the difference between the length of two consecutive words in the bad sequence $B(a)$ must be at most 1. We obtain an infinite sequence $\mathcal{S} = \mathcal{S}^{(1)}, \mathcal{S}^{(2)}, \ldots$ of words over Λ^* by concatenating bad sequences — see Fig. 1 page 11. More specifically, $\mathcal{S} = B(\mathcal{S}^{(0)}) | B(\mathcal{S}^{(t_1)}) | B(\mathcal{S}^{(t_2)}) | \ldots$ where "|" denotes the concatenation of sequences, $\mathcal{S}^{(0)} = 0$, and, for $k \geq 1$, $\mathcal{S}^{(t_k)} = (0, \ldots, 0)$, where the number of 0s is equal to the length of the last word of the bad sequence $B(\mathcal{S}^{(t_{k-1})})$, plus 1 (see Fig. 1). See also Fig. 2 for an example of construction of a bad sequence. For further references, we call these long bad *multi-diagonal* sequences.

Given the infinite sequence $\mathcal{S}$, we construct our distributed encoding (Σ, f) of the integers as follows. We set $\Sigma = \{0, 1\}^* \times \Lambda$, and the distributed code of $n \geq 1$ is $w = w_1 w_2 \ldots w_n \in \Sigma^n$ with $w_i = (\text{bin}(k), \mathcal{S}_i^{(n)})$ where $k \geq 1$ is such that the nth word $\mathcal{S}^{(n)}$ in the sequence $\mathcal{S}$ belongs to the

kth multi-diagonal sequence $B(\mathcal{S}^{(t_k)})$, and $\mathcal{S}_i^{(n)} \in \Lambda$ is the ith bit of $\mathcal{S}^{(n)}$, $i = 1, \dots, n$. For each integer $n \geq 1$ and every word $w \in \Sigma^n$, we set:

$$f(w) = \text{true} \iff \forall i \in \{1, \dots, n\}, w_i = (\text{bin}(k), \mathcal{S}_i^{(n)}) \text{ with } \mathcal{S}^{(n)} \in B(\mathcal{S}^{(t_k)}).$$

This is a correct distributed encoding since, for every integer $n \geq 1$, there exists a word $w \in \Sigma^n$ such that $f(w) = \textit{true}$, and, for every subword w' of w, $f(w') = \textit{false}$. The latter holds because every subword w' must be of the form $w' = (w_{i_j})_{j=1,\dots,m}$ with $i_j < i_{j+1}$ for $j \in [1, m)$, and if the mth element $\mathcal{S}^{(m)}$ in the sequence $\mathcal{S}$ satisfies $\mathcal{S}^{(m)} \leq_* \mathcal{S}^{(n)}$, then it cannot be the case that $\mathcal{S}^{(m)} \in B(\mathcal{S}^{(t_k)})$ too. Indeed, by construction, $B(\mathcal{S}^{(t_k)})$ is a bad sequence.

The detailed proof of Theorem 2.2 comes next.

Proof Theorem 2.2 Our distributed encoding relies on an infinite sequence $\mathcal{S} = \mathcal{S}^{(1)}, \mathcal{S}^{(2)}, \dots$ of binary words of increasing length. The ith word $\mathcal{S}^{(i)}$ of the sequence is of length i. $\mathcal{S}$ is in turn partitioned into a sequence of finite sub-sequences called *multi-diagonal sequences.* (The reader may consult Fig. 1 that depicts the first elements of $\mathcal{S}$). For every $i \geq 1$, the ith multi-diagonal sequence in $\mathcal{S}$ consists in the sub-sequence $M = \mathcal{S}^{(F^{(i-1)}(1))}, \dots, \mathcal{S}^{(F^{(i)}(1)-1)}$ of $\mathcal{S}$. Moreover, each multi-diagonal sequence in $\mathcal{S}$ satisfies the following property:

Property ($\star$): For every $i \geq 1$, and for any two words w, w' in the ith multi-diagonal sequence M in $\mathcal{S}$, w is not a subword of w'.

Before providing complete description of $\mathcal{S}$, and the proof of Property ($\star$), let us assume that we have the sequence $\mathcal{S}$ at hand, satisfying Property ($\star$). Then, we can define the distributed encoding of any integer $n \geq 1$ as follows. We set

$$\Sigma = \{0, 1\}^* \times \{0, 1\}$$

and the distributed encoding of n is the word

$$w = (x, b_1), (x, b_2), \dots, (x, b_n)$$

where $b_1 b_2 \dots b_n = \mathcal{S}^{(n)}$ is the nth binary word of the sequence $\mathcal{S}$, and x is the binary representation of the index i of the multi-diagonal sequence to which $\mathcal{S}^{(n)}$ belongs. That is, if $b_1 b_2 \dots b_n$ belongs to the ith sub-sequence, then $x = \text{bin}(i)$. Since the ith multi-diagonal sequence consists in the sub-sequence $\mathcal{S}^{(F^{(i-1)}(1))}, \dots, \mathcal{S}^{(F^{(i)}(1)-1)}$, where $i = \alpha(n)$, we get that each letter $(\text{bin}(i), b_j)$ of w, for $j = 1, \dots, n$, is on $O(\log \alpha(n))$ bits, as desired. Also, we define $f : \Sigma^* \to \{\textit{true}, \textit{false}\}$ as follows. For every word

$$w = (a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$$

in Σ^*, we set

$$f(w) = \text{true} \iff \begin{cases} \mathcal{S}^{(n)} = b_1 b_2 \dots b_n \\ a_1 = a_2 = \dots = a_n \\ \mathcal{S}^{(n)} \in i\text{th multi-diagonal sequence} \implies a_1 = \text{bin}(i) \end{cases}$$

By definition, f satisfies that, for every positive integer n, the distributed encoding w of n satisfies $f(w) = \text{true}$. On the other hand, let

$$w' = (\text{bin}(i), \mathcal{S}_{j_1}^{(n)}), (\text{bin}(i), \mathcal{S}_{j_2}^{(n)}), \dots, (\text{bin}(i), \mathcal{S}_{j_k}^{(n)})$$

be a sub-word of w. By Property $(\star)$, if

$$\mathcal{S}^{(k)} = \mathcal{S}^{(n)}_{i_1}, \mathcal{S}^{(n)}_{i_2}, \ldots, \mathcal{S}^{(n)}_{i_k}$$

then we necessarily have that $\mathcal{S}^{(k)}$ is not in the ith multi-diagonal sequence, since w' is a sub-word of w. If $\mathcal{S}^{(k)}$ belongs to the i'th multi-diagonal sequence of $\mathcal{S}$, with $i' \neq i$, then $\mathrm{bin}(i') \neq \mathrm{bin}(i)$, and therefore $f(w') = $ false, as desired. As a consequence, all the conditions of Definition 2.1 are satisfied. Therefore (Σ, f) is a distributed encoding of the integers.

Now, it remains to describe the multi-diagonal sequences so that Property $(\star)$ is satisfied. For that purpose, we review some basic facts about the vector spaces $\mathbb{N}^k, k > 0$. Let $\leq_{\text{lex}}$ denote the lexicographic order over vectors of $\mathbb{N}^k$. That is, for every $x, x' \in \mathbb{N}^k$,

$$x <_{\text{lex}} x' \iff \exists j \in [1,k] \ : \ \begin{cases} x_i = x'_i & \text{for every } i \in (j,k] \text{, and} \\ x_j < x'_j \end{cases}$$

By definition, a *bad sequence over* $\mathbb{N}^k$ is a sequence $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ of vectors of $\mathbb{N}^k$ such that for every i, j with $1 \leq i < j \leq m$, we have $x^{(i)} \not\leq_{\text{lex}} x^{(j)}$, i.e., $x^{(j)} <_{\text{lex}} x^{(i)}$ (since $<_{\text{lex}}$ is a total order). We encode the vectors $x \in \mathbb{N}^k, k > 0$, by words in $\{0,1\}^*$ as follows. Let w_x denote the encoding of the vector $x = x_1, \ldots, x_k \in \mathbb{N}^k$. We set

$$w_x = \underbrace{1\ldots1}_{x_1}0\underbrace{1\ldots1}_{x_2}0\ldots\ldots0\underbrace{1\ldots1}_{x_k}0$$

That is, each coordinate x_i of x is represented by its unary encoding, and is followed by a 0. The size $s(x)$ of the encoding of x is thus:

$$s(x) = k + \sum_{i=1}^{k} x_i.$$

Given a vector $x \in \mathbb{N}^k$, and $\mu \geq 0$, we will be interested in binary words $c_{x,\mu}$ of size $s(x) + \mu$ that consist in the word w_x followed by μ consecutive 1s, i.e.,

$$c_{x,\mu} = \underbrace{1\ldots10\ldots01\ldots10}_{w_x}\underbrace{1\ldots\ldots1}_{\mu}$$

Claim 2.3. *Let μ, μ', k, k' be positive integers, and let x, x' be two vectors in $\mathbb{N}^k$ and $\mathbb{N}^{k'}$, respectively. If either $k' < k$, or $k' = k$ with $x' <_{lex} x$, then the word $w = c_{x,\mu}$ is not a sub-word of $w' = c_{x',\mu'}$.*

Proof. In the first case, w contains strictly more 0s than w' and thus cannot be a sub-word of w'. In the second case, since $x' <_{\text{lex}} x$, there is a coordinate i, $1 \leq i \leq k$ such that $x'_i < x_i$. Hence, there are strictly more 1s between the $(i-1)$th 0 and the ith 0 of w than between the $(i-1)$th 0 and the ith 0 of w' (or more 1s at the beginning of w than at the beginning of w', if $i = 1$). Moreover, w and w' contains the same number of 0s, namely $k = k'$. For w to be a sub-word of w', the k 0s of w should be in one-to-one correspondence with the k 0s of w', and, since w has x_i consecutive 1s between its $(i-1)$th 0 and ith 0 while w' has only $x'_i < x_i$ consecutive 1s between its $(i-1)$th 0 and ith 0, there are not enough 1s in w' to accommodate the 1s in w. It follows that w cannot be a sub-word of w'. $\diamond$

As said before, $\mathcal{S}$ consists in an infinite sequence of multi-diagonal sub-sequences whose description follows. Let us fix $n \geq 1$. The multi-diagonal sequence $M = M(n)$ starts with the binary word

$$M^{(1)} = \underbrace{0\ldots\ldots0}_{n}$$

formed of n consecutive 0s, and ends with the binary word

$$M^{(\ell_n - n+1)} = \underbrace{1\ldots\ldots1}_{\ell_n}$$

formed of ℓ_n consecutive 1s, where the definition of ℓ_n follows. Note that $M^{(1)} = c_{0^n,0}$ where $0^n = (0,\ldots,0)$ is the zero vector of $\mathbb{N}^n$. Each successive word $M^{(i)}, 1 \leq i < \ell_n$, is of the form

$$M^{(i)} = c_{x^{(i)},\mu_i}$$

where $x^{(i)} \in \mathbb{N}^k$ for some $k \geq 1$, and $\mu_i \geq 0$. More precisely, $M^{(i+1)}$ is obtained inductively from $M^{(i)}$ by applying one of the following two operations to $M^{(i)}$ (see Fig. 2 for an example of that construction):

projection: If $x^{(i)} = 0^k = (0,\ldots,0)$ is the zero vector of $\mathbb{N}^k$ for some $k \geq 1$, then, for $k > 1$, we set $M^{(i+1)} = c_{x^{(i+1)},0}$ where $x^{(i+1)} \in \mathbb{N}^{k-1}$ is the vector $(0,\ldots,0,\mu_i+2)$, and, for $k=1$, we set $M^{(i+1)} = 1\ldots1$ with μ_i+2 consecutive 1s, and it is the last word of the sequence M.

maximum: If $x^{(i)}$ is a non-zero vector in $\mathbb{N}^k$ for some $k \geq 1$, then let ℓ be the leftmost non-zero coordinate of $x^{(i)}$. If $\ell = 1$, then we set $M^{(i+1)} = c_{x^{(i+1)},\mu_i+2}$ where

$$x^{(i+1)} = (x^{(i)}_1 - 1, x^{(i)}_2, \ldots, x^{(i)}_k).$$

Otherwise, we set $M^{(i+1)} = c_{x^{(i+1)},0}$ where

$$x^{(i+1)} = (0,\ldots,0,\mu_i+2, x^{(i)}_\ell - 1, x^{(i)}_{\ell+1}, \ldots, x^{(i)}_k).$$

In other words,

$$x^{(i+1)} = \max\{x \in \mathbb{N}^k : x <_{\text{lex}} x^{(i)} \text{ and } s(x) \leq s(x^{(i)}) + \mu_i + 1\}.$$

That is, $x^{(i+1)}$ is the largest vector strictly smaller than x_i in the lexicographic order, and whose size of its binary encoding is at most $|M^{(i)}| + 1$.

We note that $|M^{(1)}| = n$, and, for any i with $1 \leq i \leq n + \ell_n - 1$, we have $|M^{(i)}| = n + i - 1$. We now show that $M = M(n)$ satisfies Property $(\star)$:

Claim 2.4. *For every integer $n \geq 1$, and for any two words $w \neq w'$ in the sequence M, w is not a sub-word of w'.*

Proof. Let $w, w', |w| < |w'|$ be two words in the sequence M. If w' is the last word of M, then it contains no 0s, whereas w contains at least one 0. Hence w cannot be a sub-word of w'. Let us assume that w' is not the last word of M. Hence, $w = c_{x,\mu}$ and $w' = c_{x',\mu'}$ with $x \in \mathbb{N}^k$ and

$x' \in \mathbb{N}^{k'}$ for some integers $k, k' \geq 1$. By definition, w' is derived from w by applying a sequence of maximum and projection operations. Hence $k' \leq k$. If $k' < k$, that is, if the sequence of operations leading from w to w' contains at least one projection operation, then w is not a sub-word of w' by Claim 2.3. Otherwise, w' is derived from w by a sequence of maximum operations. Hence $x' <_{\text{lex}} x$, and it follows from Claim 2.3 that w is not a sub-word of w'. $\diamond$

Let us now turn our attention to the length of the sequence M, or, equivalently, to the length of it last word.

Claim 2.5. *The length of the last word of the sequence $M = M(n)$ is $\ell_n = F(n) - 1$.*

Proof. Let $v \geq 0$ and $k > 0$ be two integers, and let $X_{k,v}$ be the maximal sequence of binary words defined as follows :

- $X_{k,v}^{(1)} = \underbrace{0\ldots0}_{k-1}\underbrace{1\ldots1}_{v}0 = c_{x,0}$ with $x = (0, \ldots, 0, v) \in \mathbb{N}^k$
- For each $i \geq 1$, $X_{k,v}^{(i+1)}$ is the result of the maximum operation applied to $X_{k,v}^{(i)}$. If it is not possible to apply a maximum operation to $X_{k,v}^{(i)}$, then the sequence $X_{k,v}$ ends.

Note that the last word in the sequence $X_{k,v}$ is of the form

$$\underbrace{0\ldots0}_{k}\underbrace{1\ldots1}_{\mu(k,v)} = c_{0^k,\mu(k,v)}$$

where we denote by $\mu(k, v)$ the number of consecutive 1s at the end of the last word of $X_{k,v}$. We show that $\mu(k, v) = A_k(v) - 2$ for every k, by induction on k.

For $k = 1$, the sequence $X_{1,v}$ starts with the word consisting in v 1s followed by a 0. Each successive word is obtained from its predecessor by removing one of the leading 1s, and by adding two more 1s at the end. Therefore, the last word of $X_{1,v}$ consists in an initial 0 followed by $2v$ 1s. Hence, $\mu(1, v) = 2v = A_1(v) - 2$.

For the induction step, let $k \geq 2$, and assume that for every $v \geq 0$, and every $k', 1 \leq k' < k$, $\mu(k', v) = A_{k'}(v) - 2$. Let $v \geq 0$. Recall that each word $X_{k,v}^{(i)}$ is of the form $c_{x^{(i)},\mu_i}$ where $x^{(i)} \in \mathbb{N}^k$, and $\mu_i \geq 0$. Hence, we can view the sequence $X_{k,v}$ as a sequence of pairs

$$(x^{(1)}, \mu_1), (x^{(2)}, \mu_2), \ldots, (x^{(i)}, \mu_i), \ldots$$

where each $x^{(i)}$ is a vector in $\mathbb{N}^k$, and μ_i is an integer. Moreover, for every i, we have $x^{(i+1)} <_{\text{lex}} x^{(i)}$.

Let $u, 0 \leq u \leq v$. Let ℓ and m be respectively the smallest and the largest integer i such that the last coordinate of $x^{(i)}$ is u. If $v = u$, then we have $\ell = m = 1$. Otherwise, as $X_{k,v}^{(\ell)}$ is obtained by applying the maximum operation to $X_{k,v}^{(\ell-1)}$, we get that $x^{(\ell-1)} = (0, \ldots, 0, u + 1) \in \mathbb{N}^k$, and thus $x^{(\ell)} = (0, \ldots, \mu_{\ell-1} + 2, u)$, and $\mu_\ell = 0$. Similarly, applying the maximum operation to $x^{(m)}$ changes the last coordinate from u to $u - 1$. Therefore, we get $x^{(m)} = (0, \ldots, 0, u)$.

Let $y^{(\ell)}, y^{(\ell+1)}, \ldots, y^{(m)}$, with $y^{(\ell)} = (0, \ldots, 0, \mu_{\ell-1} + 2)$ and $y^{(m)} = (0, \ldots, 0, 0)$ be the vectors in $\mathbb{N}^{k-1}$ obtained by projecting out the last coordinate of $x^{(\ell)}, x^{(\ell+1)}, \ldots, x^{(m)}$, respectively. As $\mu_\ell = 0$, those vectors are the same as the vectors in the sequence $X_{(k-1,\mu_{\ell-1}+2)}$, and μ_m is thus the number of consecutive ending 1s in the last word of $X_{(k-1,\mu_{\ell-1}+2)}$. By the induction hypothesis, it thus follows that $\mu_m = \mu(k - 1, \mu_{\ell-1} + 2) = A_{k-1}(\mu_{\ell-1} + 2) - 2$.

The sequence $X_{k,v}$ can be partitioned in $v+1$ sub-sequences, called *sections*, $\sigma_v, \dots, \sigma_0$. For each u with $0 \leq u \leq v$, σ_u is the sub-sequence of $X_{k,v}$ that consists in the words whose associated vector has its last coordinate equal to u. Let us denote by t_u the number of ending 1s in the last word of σ_u (this word is $c_{(0,\dots,0,u),t_u}$). As shown above, we have $t_u = \mu(k-1, t_{u+1}+2) = A_{k-1}(t_{u+1}+2) - 2$ for every $v > u \geq 0$. Hence,

$$t_0 = \underbrace{A_{k-1}(\dots A_{k-1}(t_v+2)\dots)}_{v} - 2 = A_{k-1}^{(v+1)}(0) - 2 = A_k(v) - 2$$

The first equality follows from the facts that $t_v = 0$ and $A_{k-1}(0) = 2$. The last equality is by definition of $A_k(v)$ (cf. Eq. 1). This complete the induction.

Finally, observe that the last word of $X_{k,v}$ is the last word of σ_0. Therefore, $\mu(k,v) = t_0 = A_k(0)$. To conclude, we observe that

$$M = M(n) = X_{n,v_n}, X_{n-1,v_{n-1}}, \dots, X_{1,v_1}, \underbrace{1\dots1}_{\ell_n}$$

where $v_n = 0$, $v_i = \mu(i+1, v_{i+1}) + 2 = A_{i+1}(v_{i+2})$ for every i, $1 \leq i < n$, and $\ell_n - 2$ is the number of consecutive ending 1s in the last word of X_{1,v_1}. Hence,

$$\ell_n = \mu(1, v_1) + 2 = A_1(v_1) = A_1(A_2(v_2)) = A_1(A_2(\dots A_{n-1}(A_n(0))\dots)) = F(n) - 1.$$

This completes the proof of Claim 2.5. $\diamond$

The infinite sequence $\mathcal{S}$ is the sequence of multi-diagonal sequences

$$\mathcal{S} = M(F^0(1)), M(F^{(1)}(1)), M(F^{(2)}(1)), \dots \tag{3}$$

Let us observe that for each $i \geq 1$, the multi-diagonal sequence $M(F^0(i-1))$ starts with the word that consists in $F^{(i-1)}(1)$ 0s, and ends with the word that consists in $F^i(1) - 1$ consecutive 1s (cf. Claim 2.5). Moreover, for any two consecutive words w, w' in $M(F^0(i-1))$, we have $|w| + 1 = |w'|$. Therefore, for any j, the length of the jth word $\mathcal{S}^{(j)}$ of $\mathcal{S}$ is j. Finally, for each $j \geq 1$, consider the sub-sequence $M = \mathcal{S}^{(F^{(j-1)}(1))}, \dots, \mathcal{S}^{(F^{(j)}(1)-1)}$, that is, $M = M(F^{(j-1)}(1))$. It follows from Claim 2.4 that no two distinct words of M are sub-word of each other. Thus $\mathcal{S}$ satisfies Property ($\star$), which completes the proof of Theorem 2.2.

3 Distributed Decision

In this section, we present the application of distributed encoding of the integers to *distributed decision.* First, we describe the computational model (more details can be found in e.g. [2, 17]), and then we formally define the notions of distributed languages and decision (based on the framework of [12, 13, 14]).

Computational model We consider the standard *asynchronous wait-free read/write shared memory* model. Each process runs at its own speed, that may vary along with time, and the processes may fail by *crashing* (i.e., halt and never recover). We consider the *wait-free* model [2] in which any number of processes may crash in an execution. The processes communicate through a

$$
\begin{array}{rcl}
\mathcal{S}^{(1)} & = & 0 \text{ (1st bad sequence starts)} \\
\mathcal{S}^{(2)} & = & 11 \text{ (1st bad sequence ends)} \\
\mathcal{S}^{(3)} & = & 000 \text{ (2nd bad sequence starts)} \\
\mathcal{S}^{(4)} & = & 0110 \\
\mathcal{S}^{(5)} & = & 11010 \\
\mathcal{S}^{(6)} & = & 101011 \\
\mathcal{S}^{(7)} & = & 0101111 \\
\mathcal{S}^{(8)} & = & 11111100 \\
\mathcal{S}^{(9)} & = & 111110011 \\
\mathcal{S}^{(10)} & = & 1111001111 \\
\mathcal{S}^{(11)} & = & 11100111111 \\
\mathcal{S}^{(12)} & = & 110011111111 \\
\mathcal{S}^{(13)} & = & 1001111111111 \\
\mathcal{S}^{(14)} & = & 00111111111111 \\
\mathcal{S}^{(15)} & = & 111111111111110 \\
\mathcal{S}^{(16)} & = & 1111111111111011 \\
\mathcal{S}^{(17)} & = & 11111111111101111 \\
\mathcal{S}^{(18)} & = & 111111111110111111 \\
\vdots & \vdots & \qquad\vdots \\
\mathcal{S}^{(29)} & = & 01111111111111111111111111111 \\
\mathcal{S}^{(30)} & = & 111111111111111111111111111111 \text{ (2nd bad sequence ends)} \\
\mathcal{S}^{(31)} & = & 0000000000000000000000000000000 \text{ (3rd bad sequence starts)} \\
\mathcal{S}^{(32)} & = & 00000000000000000000000000000110 \\
\mathcal{S}^{(33)} & = & 000000000000000000000000000011010 \\
\mathcal{S}^{(34)} & = & 0000000000000000000000000001101010 \\
\vdots & \vdots & \qquad\vdots
\end{array}
$$

Figure 1: *The beginning of the infinite sequence $\mathcal{S}$.*

			$(x^{(i)})$, μ_i	
$M^{(1)}$	=	0000	$(0,0,0,0),0$	
$M^{(2)}$	=	00110	$(0,0,2),0$	(projection operation)
$M^{(3)}$	=	011010	$(0,2,1),0$	(maximum operation)
$M^{(4)}$	=	1101010	$(2,1,1),0$	
$M^{(5)}$	=	10101011	$(1,1,1),2$	
$M^{(6)}$	=	010101111	$(0,1,1),4$	
$M^{(7)}$	=	1111110010	$(6,0,1),0$	
$M^{(8)}$	=	11111001011	$(5,0,1),2$	
$M^{(9)}$	=	111100101111	$(4,0,1),4$	
$M^{(10)}$	=	1110010111111	$(3,0,1),6$	
$M^{(11)}$	=	11001011111111	$(2,0,1),8$	
$M^{(12)}$	=	100101111111111	$(1,0,1),10$	
$M^{(13)}$	=	0010111111111111	$(0,0,1),12$	
$M^{(14)}$	=	01111111111111100	$(0,14,0),0$	
$M^{(15)}$	=	110111111111111100	$(2,13,0),0$	
$M^{(16)}$	=	1011111111111110011	$(1,13,0),2$	
$M^{(17)}$	=	01111111111111001111	$(0,13,0),4$	
$M^{(18)}$	=	111111011111111111100	$(6,12,0),0$	
⋮	⋮	⋮	⋮	
⋮	⋮	⋮	⋮	
⋮	⋮	⋮	⋮	
$M^{(24)}$	=	011111111111100111111111111	$(0,12,0),12$	
$M^{(25)}$	=	1111111111111101111111111100	$(14,11,0),0$	
⋮	⋮	⋮	⋮	
			$(0,0,0),A_3(2)-2$	
			$(0,A_3(2)),0$	(projection operation)
⋮	⋮	⋮	⋮	
			$(0,0),A_2(A_3(2))-2$	
			$(A_2(A_3(2))),0$	(projection operation)
⋮	⋮	⋮	⋮	
⋮	⋮	⋮	⋮	
$M^{(F(4)-5)}$	=	01111111111.........111111111111111111	$(0),A_1(A_2(A_3(2)))-2$	
$M^{(F(4)-4)}$	=	11111111111.........1111111111111111111	$(),A_1(A_2(A_3(2)))$	(projection operation)

Figure 2: *The beginning of a long bad (multi-diagonal) sequence starting at* 0000. *Note that* $A_4(0)=2$, *and thus* $A_1(A_2(A_3(2)))=F(4)-1$.

shared memory composed of atomic registers. We associate each process p to a positive integer, its *identity* $\mathrm{id}(p)$, and the registers are organized as an array of single-writer/multiple-reader (SWMR) registers, one per process. A register i supports two operations: read() that returns the value stored in the register, and can be executed by any process, and write(x) that writes the value x in the register, and can be executed only by process with ID i. For simplicity, we use a *snapshot* operation by which a process can read all registers, in such a way that a snapshot returns a copy of all the values that were simultaneously present in the shared memory at some point during the execution of the operation. We may assume snapshots are available because they can be implemented by a wait-free algorithm using only the array of SWMR registers [1].

Distributed languages A correctness specification that is to be monitored is stated in terms of a *distributed language*. Suppose a set of processes $\{\mathrm{id}_1, \ldots, \mathrm{id}_k\} \subseteq [n]$ observe the system, and get samples $\{a_1, \ldots, a_k\}$, respectively, over a domain A. A distributed language $\mathcal{L}$ specifies whether $s = \{(\mathrm{id}_1, a_1), \ldots, (\mathrm{id}_k, a_k)\}$ corresponds to a *legal* or an *illegal* system behavior. Such a set s consisting of pairs of processes and samples is called an *instance*, and a distributed language $\mathcal{L}$ is simply the set of all legal instances of the underlying system, over a domain A of possible samples.

Given a language $\mathcal{L}$, we say that an instance s is *legal* if $s \in \mathcal{L}$ and *illegal* otherwise[3]. Given an instance $s = \{(\mathrm{id}_1, a_1), \ldots, (\mathrm{id}_k, a_k)\}$ let $\mathrm{ID}(s) = \{\mathrm{id}_1, \ldots, \mathrm{id}_k\}$ the set of identities in s and $\mathit{val}(s)$ the multiset of values in s[4].

Each process $i \in [n]$ has a read-only variable, input_i, initially equal to a symbol $\perp$ (not in A), and where the process sample a_i is deposited. We consider only the simplest scenario, where these variables change only once, from the value $\perp$, to a value in A, and this is the first thing that happens when the process starts running. The goal is for the processes to decide that, collectively, the values deposited in these variables are correct: after communicating with each other, processes output opinions. Each process i eventually deposits its opinion in its write-once variable output_i. Due to failures, it may be the case that only a subset of processes $P \subseteq [n]$ participate. The *instance* of such an *execution* is $s = \{(\mathrm{id}_i, a_i) \mid \mathrm{id}_i \in P\}$ and we consider only all executions where all processes in P run to completion (the others do not take any steps), and each one produces an opinion $u_i \in U$, where U is a set of possible opinions.[5]

Examples of a distributed language Suppose in the underlying system there is information about no more than q clients, each one identified by an integer in $[q]$. A monitor process id_i takes a sample to find out about some subset of clients $p_i \subseteq [q]$, and some subset of leaders $\ell_i \subseteq [q]$. The language k-`leader` over alphabet $A = 2^{[q]} \times 2^{[q]}$ is defined by instances s as follows. An instance $s = \Big\{(\mathrm{id}_1, (p_1, \ell_1)), \ldots, (\mathrm{id}_k, (p_k, \ell_k))\Big\}$ is in k-`leader` if and only if each known leader l belonging to some ℓ_i is a known client $p = l$ belonging to some $p \in p_j$, and the set of known leaders is of size at most k, namely

1. Validity holds: $\bigcup_{1 \le i \le k} \ell_i \subseteq \bigcup_{1 \le i \le k} p_i$, and

[3] Throughout the paper, we always assume that a process can test membership in $\mathcal{L}$, i.e., the language is sequentially decidable, in the usual sense of Turing Machine computability theory.

[4] Formally, a function that assigns to each $a \in A$ a non-negative integer specifying the number of times a is equal to one of the a_i in s.

[5] Note that in the wait-free model, a process never knows what the participating set is. Indeed, for wait-free task solvability, it is enough to consider only executions where all participating processes terminate e.g. [17].

2. Agreement holds: $|\bigcup_{1\leq i\leq k} \ell_i| \leq k$

The language `leader` corresponds to the case of $k = 1$, where a single leader should exists, and this leader should be a known client.

A simplified version of `leader` is when it is always the case that that the sample of process i, (p_i, ℓ_i), always consists of $p_i = \{\mathrm{id}_i\}$, for every process i. Then we have the language *leader election*, for which it is required that one unique process be identified as the leader by all the other processes. This requirement is captured by the language `leader` defined over $A = [n]$ as follows:

$$s = \big\{(\mathrm{id}_1, \ell_1), \dots, (\mathrm{id}_k, \ell_k)\big\} \in \texttt{leader} \iff \exists i \in [k] : \mathrm{id}_i = \ell_1 = \dots = \ell_k. \tag{4}$$

An instance is legal if and only if all the processes agree on the identity ℓ of one of them.

Deciding a distributed language Deciding a language $\mathcal{L}$ involves two components: an *opinion-maker* M, and an *interpretation* μ. The opinion-maker is the distributed algorithm executed by the processes. Each process produces an individual *opinion* in U about the legality of the global instance. The interpretation μ specifies the way one should interpret the collection of individual opinions produced by the processes. It guarantees the minimal requirement that the opinions of the processes should be able to distinguish legal instances from illegal ones according to $\mathcal{L}$. Consider the set of all multi-sets over U, each one with at most n elements. Then $\mu = (\mathbf{Y}, \mathbf{N})$ is a partition of this set. $\mathbf{Y}$ is called the "yes" set, and $\mathbf{N}$ is called the "no" set.

For instance, when $U = \{0, 1\}$, process may produce as an opinion either 0 or 1. Together, the monitors produce a multi-set of at most n boolean values. We do not consider which process produce which opinion, but we do consider how many processes produce a given opinion. The partition produced by the AND-operator [13] is as follows. For every multi-set of opinions S, set $S \in \mathbf{Y}$ if every opinion in S is 1, otherwise, $S \in \mathbf{N}$.

Given a language $\mathcal{L}$ over an alphabet A, a *distributed monitor for* $\mathcal{L}$ is a pair (M, μ), an opinion maker M and an interpretation μ, satisfying the following, for every execution E of M starting with instance $s = \{(\mathrm{id}_i, a_i) \mid \mathrm{id}_i \in P\}$, $P \subseteq [n]$.

Definition 3.1. *The pair (M, μ) decides $\mathcal{L}$ with opinions U if every execution E on instance $s = \{(\mathrm{id}_i, a_i) \mid \mathrm{id}_i \in P,\ a_i \in A\}$ satisfies*

- *The input of process i is a_i, and the opinion-maker M outputs on execution E an opinion $u_i \in U$.*
- *The instance $s \in \mathcal{L}$ if and only if the processes produce a multiset of opinions $S \in \mathbf{Y}$. Given that $(\mathbf{Y}, \mathbf{N})$ is a partition of the multisets over U, $s \notin \mathcal{L}$ if and only $S \notin \mathbf{Y}$.*

Non-deterministic distributed decision In the same way NP extends P, we extend the notion of distributed decision to distributed *verification*, similar to [12]. In addition to its input x_i, process id_i receives a binary string $c_i \in \{0, 1\}^*$. The set $c = \{(\mathrm{id}_i, c_i) \mid \mathrm{id}_i \in P\}$ is called a *distributed certificate* for processes P. The pair (M, μ) is a *distributed verifier* for $\mathcal{L}$ with opinions U if for any $s = \{(\mathrm{id}_i, a_i) \mid \mathrm{id}_i \in P,\ a_i \in A\}$, the following hold

1. For any certificate $c = \{(\mathrm{id}_i, c_i) \mid \mathrm{id}_i \in P\}$, the input of process i is the pair (a_i, c_i), and the opinion-maker M outputs on every execution E an opinion $u_i \in U$.

2. (a) If instance $s \in \mathcal{L}$ then there exists a certificate c such that in every execution the processes produce a multiset of opinions $S \in \mathbf{Y}$.
   (b) If instance $s \notin \mathcal{L}$ then for any certificate c the processes produce a multiset of opinions $S \in \mathbf{N}$.

Note that we do not enforce any constraints on the running time of the opinion maker M. Nevertheless, M must be *wait-free*, and must not be fooled by any "fake" certificate c for an instance $s \notin \mathcal{L}$.

4 Efficient non-deterministic decision

We first show that it is possible to verify every distributed language using three opinions, with very small size certificates. Then we show that with constant size certificates, almost constant size number of opinions are sufficient. We use the function α defined in Eq. (2) and the distributed encoding notion of Definition 2.1.

4.1 Impossibility of verifying with two opinions

Ideally, we would like to deal with opinion-makers using very few opinions (e.g., just true or false), and with simple interpreters (e.g., the boolean AND operator). However, the following result shows that even very classical languages like consensus cannot be verified with such simple verifiers.

Theorem 4.1. *There are languages that cannot be verified using only two opinions, even restricted to instances of dimension at most 2 (i.e., 3 processes), and regardless to the size of the certificates.*

The proof of Theorem 4.1 uses arguments from combinatorial topology. Indeed, it is known [17, 18] that, roughly, a task is wait-free solvable if and only if there is a simplicial map from a subdivision of its *input complex* to its *output complex*. For instance, consensus is not *wait-free solvable* because any subdivision preserves the connectivity of the consensus input complex, while the consensus output complex is disconnected, from which it follows that a simplicial map between the two complexes cannot exist. We show that *binary* consensus among three processes is not *wait-free verifiable* with only two opinions. This is obtained by assuming that binary consensus in wait-free verifiable, and then establishing a contradiction on the structure of the *protocol complex*.

Proof of Theorem 4.1 We explicitly describe a distributed language $\mathcal{L}$ that cannot be verified using only two opinions. This language is natural in the sense that it fits with the specification of verifying the correctness of *binary consensus*. (Recall that, in the consensus task, every process i is given an input value s_i, and must produce an output value t_i such that all output values are identical, and equal to one of the input values). Also, this language is identity-oblivious, in the sense that if $\{(\mathrm{id}_1, x_1), \dots, (\mathrm{id}_n, x_n)\} \in \mathcal{L}$, then $\{(\mathrm{id}'_1, x_1), \dots, (\mathrm{id}'_n, x_n)\} \in \mathcal{L}$ for every $\mathrm{id}'_1, \dots, \mathrm{id}'_n$. We thus omit the identities, and define $\mathcal{L}$ as follows:

$$\{(s_1, t_1), \dots, (s_n, t_n)\} \in \mathcal{L} \iff \begin{cases} \forall i : (s_i, t_i) \in \{0,1\}^2 \\ \exists i : t_1 = \dots = t_n = s_i \end{cases}$$

Assume, for the purpose of contradiction that $\mathcal{L}$ can be verified with an opinion-maker M producing only two values a and b, for an appropriate setting of certificates. In particular, we can assume, w.l.o.g., that M outputs b on the legal instance $(0,0)$ provided with its appropriate certificate, and a on the illegal instance $(0,1)$. That is, the interpretation μ accepts $\{a\}$, but rejects $\{b\}$. We obtain a contradiction just by considering instances of dimension at most 2 (i.e., with at most 3 pairs (s_i,t_i)). We first show that the interpretation must accept $\{a,a\}$ and reject the multisets $\{a,b\}$, $\{b,b\}$, $\{a,a,b\}$ and $\{a,b,b\}$. We then obtain a contradiction by exhibiting an execution of M on a legal instance with the appropriate certificate in which the output is either the multiset of opinions $\{a,a,b\}$ or $\{a,b,b\}$.

Without loss of generality, we assume that the read/write wait-free protocol M consists in a certain number β of asynchronous rounds. In each round, process p writes its state in its dedicated register, takes a snapshot of the array of registers, and updates its state. We focus on a certain class of executions, called *immediate snapshot* executions [17], which can be divided into blocks. In a block, only a subset of processes is active. They first simultaneously write before taking a snapshot. Following the work in [18], we know that immediate snapshot executions can be represented by simple combinatorial objects, namely, *simplexes* and *complexes* [17].

A complex $\mathcal{K}$ is a set of vertices $V(\mathcal{K})$, and a family of finite, nonempty subsets of $V(\mathcal{K})$, called simplexes, satisfying: (1) if $v \in V(\mathcal{K})$ then $\{v\}$ is a simplex, and (2) if s is a simplex, so is every nonempty subset of s. A simplex τ is a face of a simplex σ if τ is a subset of σ. If τ is not equal to σ then τ is a proper face of σ. The complex induced by a simplex σ consists in σ and all its faces. The dimension of a simplex s is $|s|-1$. In distributed computing, a vertex represents a local state, a simplex a global state and a complex a collection of global states. Hence, one of the labels of each vertex is an identity. We denote by $\mathrm{ID}(\sigma)$ the identities of the vertexes of σ. An *input complex* $\mathcal{I}$ represents possible initial states. In this case, each vertex is additionally labeled with an input value (in our case a pair (s_i,t_i) and a certificate c_i). Each protocol solving a task has an associated *protocol complex* in which each vertex is labeled with a process id and the decision of the process (in our case an opinion a or b). If σ is an input complex, let $\mathcal{P}(\sigma)$ be the complex of final states that can be reached from the initial state σ by immediate snapshot executions of the protocol. For the wait-free model of computation, previous works (e.g., [17, 18]) have shown that $\mathcal{P}(\sigma)$ is a subdivision of σ.

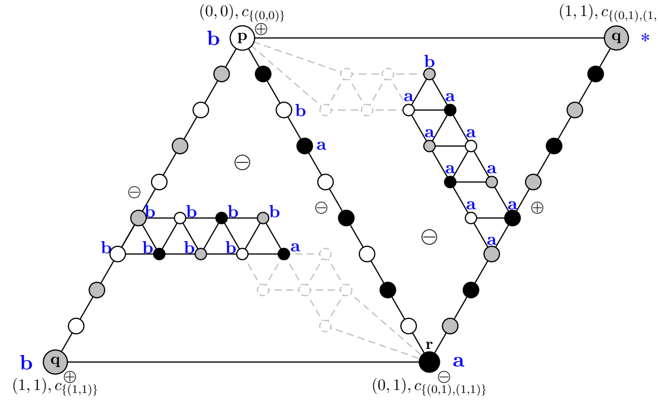

Figure 3: Part of the opinion-maker protocol complex with input simplexes $\{(p,(0,0),c_{(0,0)}),(q,(1,1),c_{(1,1)}),(r,(0,1),c_{(0,1),(1,1)})\}$ and $\{(p,(0,0),c_{(0,0)}),$ $(q,(1,1),c_{(0,1),(1,1)}),$ $(r,(0,1),c_{(0,1),(1,1)})\}$. For a legal instance s, c_s denote the appropriate certificate for s.

We first consider the protocol sub-complex depicted in Fig. 3. We establish that it includes a simplex τ of dimension 2 labeled $\{a, b, b\}$. This simplex τ is reachable from an input simplex corresponding to the instance $s = \{(0,0),(0,1),(1,1)\}$. That is, there is an execution of the opinion maker M in which the three participating processes produce the opinions a, b and b respectively and the instance in that execution is $s \notin \mathcal{L}$. Therefore, the multiset $\{a, b, b\}$ must be rejected by the interpretation μ.

Consider the two corners labeled with p (white) and r (black) at the top left and bottom right (see Fig. 3). The top corner represents a solo execution by process p with input $(0,0)$ and the appropriate certificate for this input denoted $c_{\{(0,0)\}}$. In this execution, p opinion must be b as $\{(0,0)\} \in \mathcal{L}$ and p has been provided with the right certificate. Similarly, the bottom corner represents a solo execution by process r with input $(0,1)$. As $\{(0,1)\} \notin \mathcal{L}$, r opinion must be rejected by μ, that is r opinion must be a. Each vertex in the subdivided edge linking these two corners is labeled with an opinion in $\{a, b\}$. Since the corners are labeled a and b respectively, there are two neighbors vertexes labeled a and b respectively. This means that the multiset of opinions $\{a, b\}$ is produced by the opinion maker in an execution on instance $\{(0,0),(0,1)\}$, which is not in $\mathcal{L}$. Therefore $\{a, b\}$ is rejected by μ.

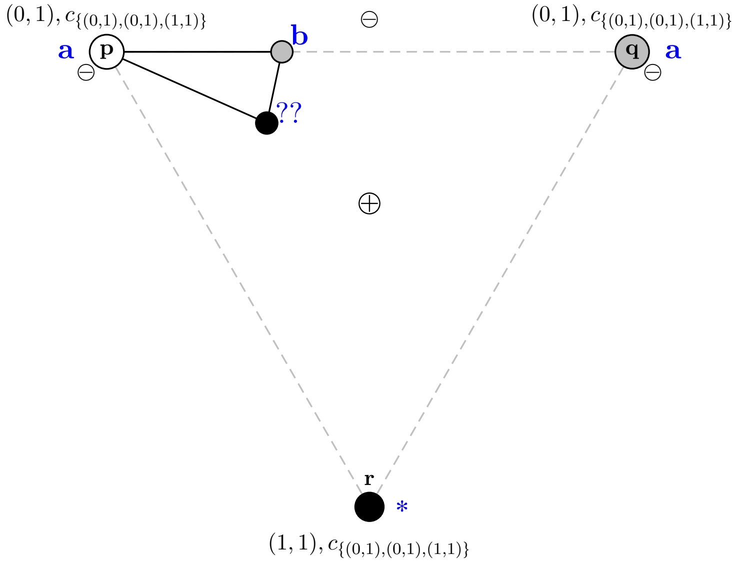
$(0,1), c_{\{(0,1),(0,1),(1,1)\}}$
a
p
$\ominus$
b
??
$\ominus$
$\oplus$
$(0,1), c_{\{(0,1),(0,1),(1,1)\}}$
q
a
$\ominus$
r
*
$(1,1), c_{\{(0,1),(0,1),(1,1)\}}$

Figure 4: Part of the opinion-maker protocol complex generated by the input simplexe $\{(p,(0,1),c_{(0,1),(0,1),(1,1)}),(q,(0,1),c_{(0,1),(0,1),(1,1)})),(r,(1,1),c_{(0,1),(0,1),(1,1)})\}$. $c_{(0,1),(0,1),(1,1)}$ is the certificate associated with the instance $\{(0,1),(0,1),(1,1)\}$.

Let us now concentrate on the subdivided edge with extremities r and q on the right triangle (see Fig. 3). Note that no simplex on this edge may be labeled $\{a, b\}$ since the input instance is $s = \{(0,1),(1,1)\} \in \mathcal{L}$, the processes are provided with the appropriate certificate (denoted $c_{\{(0,1),(1,1)\}}$) and μ rejects $\{a, b\}$. As the bottom corner (colored black and labeled r in the picture) is labeled a, this implies that every vertex is labeled a. Therefore the multiset $\{a, a\}$ is accepted by μ.

Now, since the protocol sub-complex generated by the input simplex

$$\{(p, ((0,0), c_{(0,0)}), (q, ((1,1), c_{(0,1),(1,1)}), (r, ((0,1), c_{(0,1),(1,1)})))\}$$

is a subdivided simplex of that simplex, there is a sequence of 2-dimensional simplexes $\tau_1, \ldots, \tau_\ell$ such that (1) τ_1 contains the white corner p, (2) $\tau_i \cap \tau_{i+1}$ is a simplex of dimension 1, for each $i, 1 \leq i < \ell$ and, (3) τ_ℓ contains at least two vertexes labeled a. As every vertex is labeled a or b, it follows that there is a 2-dimensional simplex τ_i in the sequence which is labeled a, a, b. This simplex represents on execution of the opinion maker on instance $s = \{(0,0), (1,1), (0,1)\} \notin \mathcal{L}$. Therefore the multiset $\{a, a, b\}$ must be rejected by μ.

Let us now concentrate on the subdivided triangle on the left, which is the protocol sub-complex generated by the input simplex

$$\{(p, ((0,0), c_{(0,0)}), (q, ((1,1), c_{(1,1)}), (r, ((0,1), c_{(0,1),(1,1)})))\}.$$

The bottom left corner q is labeled b, as it corresponds to a solo execution by process q with the right certificate on an instance, namely $\{(1,1)\}$ in $\mathcal{L}$. Each simplex on the subdivided edge joining the corners q and p is labeled a, b or b, b. This is because $\{a, a\}$ is accepted by μ, and each of those simplexes correspond to execution on the instance $\{(0,0), (1,1)\}$ which is not in $\mathcal{L}$. By a parity argument, there must exist a simplex labeled b, b. Then, following the same reasoning as above, one can show that there is a 2-dimensional simplex $\tau'_{i'}$ labeled b, b, a. This simplex corresponds to an execution on $s = \{(0,0), (1,1), (0,1)\}$ and thus the multiset $\{a, b, b\}$ is rejected by μ.

Finally, consider the protocol sub-complex induced by the input simplex

$$\{(p, ((0,1), c_p)), (q, ((0,1), c_q)), (r, ((1,1), c_r))\}$$

depicted in Fig. 4. Here, c_p, c_q, c_r denote the appropriate certificate for processes p, q and r respectively for the instance $\{(0,1), (0,1), (1,1)\}$. Let us examine the labels of the marked output simplex on the top left corner. Let τ be that simplex. Since $\{(0,1)\} \notin \mathcal{L}$, the label of the white vertex must be a. Indeed, it corresponds to a solo execution by process p on the illegal instance $\{(0,1)\}$. The simplex formed by the gray and the white vertexes corresponds to a 2-processes execution on the illegal instance $\{(0,1), (0,1)\}$. The multiset of opinions output in that execution has thus to be rejected by μ. Therefore the label of the gray vertex must be b. Let x be the opinion associated with the black vertex. Simplex τ corresponds to an execution of the opinion maker on the legal instance $\{(0,1), (0,1), (1,1)\}$ in which the processes are provided with the right certificates. Hence, the multiset of opinions $\{a, b, x\}$ is accepted by μ. However, we have seen that both multisets $\{a, b, b\}$ and $\{a, b, a\}$ are rejected by the interpretation μ. Therefore $x \notin \{a, b\}$, a contradiction. Therefore, at least three opinions must be used by the verifier for the language corresponding to checking correctness of binary consensus on three processes.

4.2 Verification three opinions and almost constant size certificates

On the other hand, it was proved in [19] that every distributed language can be verified using only three opinions (true, false, undetermined). However, the verifier in [19] exhibited to establish this result uses certificates of size $O(\log n)$ bits for n-dimensional instances. The following shows how to improve this bound using distributed encodings and function α (Eq. (2)).

Theorem 4.2. *Every distributed language can be verified using three opinions, with certificates of size $O(\log \alpha(n))$ bits for n-process instances.*

Proof. Let $\mathcal{L}$ be a distributed language. The verifier (M, μ) for $\mathcal{L}$ is based on the distributed encoding (f, Σ) of the positive integers whose existence is established in Theorem 2.2. Given an $(n-1)$-dimensional instance $s = \big((j_1, x_{j_1}), \dots, (j_n, x_{j_n})\big) \in \mathcal{L}$, with $j_1 < j_2 < \cdots < j_n$, the certificate for s is the distributed code $c = (c_{j_1}, \dots, c_{j_n}) \in \Sigma^n$ of n. Recall that, by definition of the distributed encoding, we have $f(c) = \text{true}$, while $f(c') = \text{false}$ for every subword $c' \prec c$ of c. Algorithm 1 describes a verifier generating one out of three possible opinions (true, false, or undetermined) at each process, using such certificates.

Algorithm 1. Universal verifier with 3-valued opinions: code of the opinion maker M for process i.

```
Require: input pair value-certificate $(x_i, c_i)$
write $(i, x_i, c_i)$
$view \leftarrow$ snapshot()
let $view = \{(j_1, x_{j_1}, c_{j_1}), \dots, (j_k, x_{j_k}, c_{j_k})\}$
let $s = \big((j_1, x_{j_1}), \dots, (j_k, x_{j_k})\big)$
let $c = (c_{j_1}, \dots, c_{j_k})$
if $s \in \mathcal{L}$ then
    if $f(c)$ then
        decide true
    else
        decide undetermined
    end if
else                                              ▷ $s \notin \mathcal{L}$
    if (not $f(c)$) and ($\exists c' \prec c : f(c')$) then
        decide false
    else
        decide undetermined
    end if
end if
```

Algorithm 2. Universal verifier with 1-bit certificate: code of the opinion maker M for process i.

```
Require: input pair value-certificate $(x_i, c_i)$
write $(i, x_i, c_i)$
$view \leftarrow$ snapshot()
let $view = \{(j_1, x_{j_1}, c_{j_1}), \dots, (j_k, x_{j_k}, c_{j_k})\}$
let $s = \big((j_1, x_{j_1}), \dots, (j_k, x_{j_k})\big)$
let $c = (c_{j_1}, \dots, c_{j_k})$
if $s \in \mathcal{L}$ then
    if $c = \mathcal{S}^{(k)}$ then                        ▷ $k = |c|$
        decide $(\text{true}, \text{level}(k))$
    else
        decide $(\text{false}, \text{level}(k))$
    end if
else                                              ▷ $s \notin \mathcal{L}$
    if $\exists (s', c') : s' \subset s,\ s' \in \mathcal{L},\ c' \prec c,\ c' = \mathcal{S}^{(|c'|)}$,
                and $\text{level}(|c'|) = \text{level}(k)$) then
        decide $(\text{false}, \text{level}(k) + 1)$
    else
        decide $(\text{false}, \text{level}(k))$
    end if
end if
```

In Algorithm 1, every participating process p_i writes its identity i, its value x_i, and its certificate c_i in memory, and then takes a snapshot. The latter provides process i with a view of the system including all the data written by the k participating processes which wrote before it takes snapshot (including itself). This view enables process i to compute an instance s and a word c, ordered by the identities of the processes in its snapshot. Based on whether or not $s \in \mathcal{L}$, on whether $f(c) =$ true or false, and on whether c contains a sub-words c' such that $f(c') = \text{true}$, process i decides true, false, or undetermined, as specified in Algorithm 1.

The interpretation μ of the opinions decided by the processes uses the following truth table, where T stands for true, F for false, and U for undetermined:

$\wedge$	T	U	F
T	T	T	F
U	T	U	F
F	F	F	F

The interpreter μ accepts a multiset $\{o_1, \dots, o_n\}$ of options if and only if

$$\bigwedge_{i=1}^{n} o_i = T.$$

In other words, μ accepts if and only if no processes decide false (F), and at least one process decides true (T). That is, the yes-set consist in all multi-sets containing at least one opinion true, and no opinions false.

By construction, this verifier uses three opinions, and certificates on $O(\log \alpha(n))$ bits for n-dimensional instances. It remains to prove its correctness. We need to consider only executions in which all processes in $\mathrm{ID}(s)$ decide.

Let $s \in \mathcal{L}$ be an $(n-1)$-dimensional instance, and let c be its valid certificate. At least one process has a view equal to (s, c). Since $s \in \mathcal{L}$ and $f(c) = \text{true}$, this process decides true. Moreover, by definition of the distributed encoding of the integers, there are no sub-words c' of c such that $f(c') = \text{true}$. Thus no processes decide false. As a consequence, the interpretation of all the opinions is true, as desired.

Let $s \notin \mathcal{L}$ be an $(n-1)$-dimensional instance, and let c be any certificate. If no processes decide true, then we are done since the interpretation of all the opinions is then false, as desired. So assume that some process decides true. The view of this process in composed of an instance $s' \subset s$ accompanied by a certificate $c' \prec c$, satisfying $s' \in \mathcal{L}$ and $f(c') = \text{true}$. Since $f(c') = \text{true}$, we get that $f(c) = \text{false}$ by definition of the distributed encoding of the integers. Therefore, Instruction 13 of the opinion maker M in Algorithm 1 implies that any process with a view equal to (s, c) decides false. As a consequence, the interpretation of all the opinions is again false, as desired, which completes the proof. □

4.3 Verification with constant-size certificates

Interestingly, we can reduce the size of the certificates even further, at the cost of slightly increasing the number of opinions (to a number which remains constant for all reasonable number of processes).

Theorem 4.3. *Every language can be verified with 1-bit certificates, using $O(\alpha(n))$ opinions for n-dimensional instances.*

Proof. Let $\mathcal{L}$ be a distributed language. We describe a verifier (M, μ) for it. Given an $(n-1)$-dimensional instance $s = \big((j_1, x_{j_1}), \dots, (j_n, x_{j_n})\big) \in \mathcal{L}$, with $j_1 < j_2 < \dots < j_n$, the certificate for s is $c = (c_{j_1}, \dots, c_{j_n}) = \mathcal{S}^{(n)} \in \{0,1\}^n$, where the sequence $\mathcal{S}$ is defined in the proof of Theorem 2.2. Recall that $\mathcal{S} = M(n_0)|M(n_1)|M(n_2)|M(n_3)|\dots$ for an appropriate sequence of integers n_i, $i \geq 0$. For each i, the sub-sequence $M(n_i)$ is called multi-diagonal sequence. By construction of $\mathcal{S}$, it may happen that $\mathcal{S}^{(n)} \leq_* \mathcal{S}^{(n')}$ for two different integers n and n'. However, in this case, the proof of Theorem 2.2 states that $\mathcal{S}^{(n)}$ and $\mathcal{S}^{(n')}$ belong to two different multi-diagonal sequences $M(n_i)$ and $M(n_j)$, $j \neq i$. In accordance to this fact, we define the *level* of n as follows. If the n-th word $\mathcal{S}^{(n)}$ of the sequence $\mathcal{S}$ belongs to $M(n_i)$, for some $i \geq 0$, then we set $\text{level}(n) = i$.

Algorithm 2 describes an opinion maker M using the words of $\mathcal{S}$ as certificates, in which the processes produce opinions true or false, coupled with a “degree of confidence” corresponding to the level of n for the observed $(n-1)$-dimensional instance s. This algorithm follows the same structure as Algorithm 1. The only differences are due to the fact that, since the certificates are based on the word in $\mathcal{S}$ instead of being based on the distributed encoding of the integers, it may happen that a certificate c' for a sub-instance s' of the actual instance s fits with the dimension of that instance, that is, $c' = \mathcal{S}^{(|s'|)}$, even if the certificate c for s also satisfies $c = \mathcal{S}^{(|s|)}$. This prevents the participating processes to be completely sure of the dimension of the full instance. Therefore, the true-false decision of each process is weighted by the level of the observed instance, so that decisions taken based on a higher-dimensional instances overcome decisions taken based on lower-dimensional instances.

Specifically, the interpreter accepts a multiset $\{o_1, \dots, o_n\}$ of options, with $o_i = (b_i, \ell_i)$, $b_i \in \{\text{true}, \text{false}\}$, and $\ell_i \geq 0$, if and only if there exists i such that $b_i = \text{true}$ and, for every $j \neq i$, $\ell_j \leq \ell_i$. In other words, the interpreter accepts if and only if at least one process decides true with some confidence level ℓ, and no processes decide false with a confidence level higher than ℓ.

This verifier uses $O(\alpha(n))$ opinions for n-dimensional instance, with 1-bit certificates per process. It remains to prove its correctness. We need to consider only executions in which all n processes decide.

Let $s \in \mathcal{L}$ be an $(n-1)$-dimensional instance, and let $c = \mathcal{S}^{(n)}$ be its valid certificate. At least one process has a view equal to (s, c). Since $s \in \mathcal{L}$ and $c = \mathcal{S}^{(n)}$, this process decides $(\text{true}, \text{level}(n))$. Since the input instance s has dimension $n-1$, no processes can decide (false, ℓ) with $\ell > \text{level}(n)$. Thus the interpretation of all the opinions is true, as desired.

Let $s \notin \mathcal{L}$ be an $(n-1)$-dimensional instance, and let c be any certificate. If no processes decide true, then we are done since the interpretation of all the opinions is then false, as desired. So assume that some process decides true, with some level of confidence ℓ. The view of this process in composed of a instance $s' \subset s$ accompanied by a certificate $c' \prec c$, satisfying $s' \in \mathcal{L}$ and $c' = \mathcal{S}^{(|c'|)}$. If $\ell < \text{level}(n)$, then, since any process with a view equal to (s, c) decides (false, ℓ') with $\ell' \geq \text{level}(n)$, the interpretation of all the opinions is false, as desired. If $\ell = \text{level}(n)$, then Instruction 13 implies that any process with a view equal to (s, c) decides $(\text{false}, \text{level}(n) + 1)$. As a consequence, since $\ell \leq \text{level}(n)$, the interpretation of all the opinions is again false, as desired, which completes the proof. □

Acknowledgment: The third author is thankful to Philippe Duchon and Patrick Dehornoy for fruitful discussions on wqos.

References

[1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, Nir Shavit: Atomic Snapshots of Shared Memory. *J. ACM* **40**(4): 873–890 (1993).

[2] Hagit Attiya, Jennifer Welch: Distributed Computing: Fundamentals, Simulations, and Advanced Topics. *John Wiley & Sons* (2004).

[3] Andreas Bauer, Martin Leucker, Christian Schallhart: Comparing LTL Semantics for Runtime Verification. *J. Log. Comput.* **20**(3): 651–674 (2010).

[4] Lélia Blin, Pierre Fraigniaud, Boaz Patt-Shamir: On Proof-Labeling Schemes versus Silent Self-stabilizing Algorithms. In proc. *16th Int. Symposium on Stabilization, Safety, and Security of Distributed Systems* (SSS), pp. 18–32, 2014.

[5] Tushar Deepak Chandra, Sam Toueg: Unreliable Failure Detectors for Reliable Distributed Systems. *J. ACM* **43**(2): 225–267 (1996).

[6] Byron Cook, Andreas Podelski, Andrey Rybalchenko: Proving program termination. *Communications of the ACM* **54**(5):88–98 (2011).

[7] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, R. Wattenhofer: Distributed Verification and Hardness of Distributed Approximation. *SIAM J. Comput.* **41**(5): 1235–1265 (2012).

[8] L. Feuilloley and P. Fraigniaud: Randomized Local Distributed Network Computing. In proc. *27th ACM Symp. on Parallelism in Algorithms and Architectures* (SPAA), pp. 340–349, 2015.

[9] D. Figueira, S. Figueira, S. Schmitz, P. Schnoebelen: Ackermannian and Primitive-Recursive Bounds with Dickson's Lemma. In proc. *26th IEEE Symp. on Logic in Computer Science* (LICS), pp. 269–278, 2011.

[10] Michael J. Fischer, Nancy A. Lynch, Mike Paterson: Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* **32**(2): 374–382 (1985).

[11] Pierre Fraigniaud, Mika Göös, Amos Korman, Merav Parter, David Peleg: Randomized Distributed Decision. *Distributed Computing* **27**(6):419–434 (2014).

[12] Pierre Fraigniaud, Amos Korman, David Peleg: Towards a Complexity Theory for Local Distributed Computing. *J. ACM* **60**(5):35 (2013).

[13] Pierre Fraigniaud, Sergio Rajsbaum, Corentin Travers: Locality and Checkability in Wait-Free Computing. *Distributed Computing* **26**(4): 223–242 (2013).

[14] P. Fraigniaud, S. Rajsbaum, C. Travers: On the Number of Opinions Needed for Fault-Tolerant Run-Time Monitoring in Distributed Systems. In proc. *5th Int. Conference on Runtime Verification* (RV), Springer, LNCS 8734, pp. 92–107, 2014.

[15] P. Fraigniaud, S. Rajsbaum, M. Roy, C. Travers: The Opinion Number of Set-Agreement. In proc. *18th Int. Conf. on Principles of Distributed Systems* (OPODIS), Springer, LNCS 8878, pp. 155–170, 2014.

[16] Mika Göös, Jukka Suomela: Locally Checkable Proofs. In proc. *30th ACM Symp. on Principles of Distributed Computing* (PODC), pp. 159–168, 2011.

[17] Maurice Herlihy, Dmitry N. Kozlov, Sergio Rajsbaum: Distributed Computing Through Combinatorial Topology. *Morgan Kaufmann*, 2013.

[18] Maurice Herlihy, Nir Shavit: The Topological Structure of Asynchronous Computability. *J. ACM* **46**(6): 858–923 (1999).

[19] Marc Jeanmougin: Checkability in Asynchronous Error-Prone Distributed Computing Using Few Values. *Master Thesis Report*, University Paris Diderot, 2013.

[20] Amos Korman, Shay Kutten, David Peleg: Proof Labeling Schemes. *Distributed Computing* **22**(4): 215–233 (2010).

[21] Joseph B. Kruskal: The Theory of Well-Quasi-Ordering: A Frequently Discovered Concept. *Journal of Combinatorial Theory A* **13**(3): 297–305 (1972).

[22] E.C. Milner: Basic WQO- and BQO-theory. In proc. *Graphs and Order, The Role of Graphs in the Theory of Ordered Sets and Its Applications.* NATO ASI Series 147, pp. 487–502, 1985.

[23] D. Peleg: Distributed Computing: A Locality-Sensitive Approach. *SIAM*, 2000.

[24] Sylvain Schmitz, Philippe Schnoebelen: Multiply-Recursive Upper Bounds with Higman's Lemma. In proc. *38th International Colloquium on Automata, Languages and Programming* (ICALP), Springer, LNCS 6756, pp. 441–452, 2011.

[25] Sylvain Schmitz, Philippe Schnoebelen: Algorithmic Aspects of WQO Theory. *Tech. Report* Hal Archive, 2013.

[26] Philippe Schnoebelen: Verifying Lossy Channel Systems has Nonprimitive Recursive Complexity. *Inf. Process. Lett.* **83**(5): 251–261 (2002).

[27] Philippe Schnoebelen: Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets. In Proc. *35th Int. Symp. on Mathematical Foundations of Computer Science* (MFCS), Springer, LNCS 6281, pp. 616–628, 2010.

[28] Alan Turing: Checking a Large Routine. In Report of a *Conference on High Speed Automatic Calculating Machines*, pp. 67–69, 1949.