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

Pierre Fraigniaud[1], Sergio Rajsbaum[2], and Corentin Travers[3]

[1] Univ. Paris Diderot and CRNS, France,
pierre.fraigniaud@liafa.univ-paris-diderot.fr
[2] Instituto de Matemáticas, UNAM, Mexico, rajsbaum@im.unam.mx
[3] 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.

In this paper we study non-deterministic distributed decision in systems where asynchronous processes may crash. 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 $\lceil \log \alpha(n) \rceil + 1$ 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* $(\{0,1\}^*, \leq_*)$ 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: runtime verification, distributed decision, distributed verification, well-quasi-ordering, wait-free computing, combinatorial topology.

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

1 Introduction

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 [11,14], designing time lower bounds on the hardness of distributed approximation [7], estimating the complexity of logics required for distributed run-time verification [13], and elaborating a distributed computing complexity theory [10,15].

The predicate to be decided in a distributed decision problem is specified as the set of all valid input vectors, called a *distributed language* $\mathcal{L}$. Each process p_i is given an input value x_i, and should produce an output value $o_i \in U$, where U is the set of possible *opinions*. The processes should collectively decide whether their vector 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 vector of opinions $o = (o_i)_i$ emitted by the processes.

In a distributed system where n 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 [9]). Often processes emit two possible opinions, $U = \{true, false\}$, and the collective opinion is interpreted as the conjunction of the emitted values. Some languages $\mathcal{L}$ may be decided by emitting only two opinions, but not all. In fact, it is known that up to n different opinions may be necessary to decide some languages [13], irrespectively of how the opinions are interpreted. For example, for the *k-set agreement* language, specifying that at most k leaders are elected, a set U of $\min\{2k, n\}+1$ opinions is necessary and sufficient in a system where n asynchronous processes may crash [14]. A measure of the complexity of $\mathcal{L}$ is the minimum number of opinions needed to decide it.

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 vector x is 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 opinions emitted by the processes.

This non-deterministic framework is inspired by classical complexity theory, but it has been used before also in various distributed settings, e.g. in distributed complexity [10], in silent self-stabilization [4] (as captured by the concept of proof-labeling schemes [19]), as well as failure detectors [5] where an underlying layer produces 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, it is natural to seek certificates that are as small as possible, perhaps for information theoretic purposes, privacy purposes, or because certificates have to be exchanged among processes [4,19]. 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[4].

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

We define a distributed encoding of the (natural) integers as a collection of code-words providing every integer n with a code $w = (w_i)_{i=1,\ldots,n}$ in Σ^n, where Σ is a (possibly infinite) alphabet, such that, for any $k \in [1, n)$, no subwords[5] $w' \in \Sigma^k$ of w is encoding k. Trivially, every integer $n \geq 1$ can be (distributedly) encoded by the word $w = (\mathrm{bin}(n), \ldots, \mathrm{bin}(n)) \in \Sigma^n$ with $\Sigma = \{0,1\}^*$, where $\mathrm{bin}(n)$ 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 of $\lceil \log \alpha(n) \rceil + 1$ 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

[4] 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].

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

long (so-called) *bad sequences* of words over $(\Lambda^*, \leq_*)$ starting from any word $a \in \Lambda^*$, 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 second result is an application of distributed encoding of the integers to distributed computing. This is a novel use of well-quasi-orderings, that lies outside the traditional applications to logic and complexity theory. Specifically, we prove that any distributed language $\mathcal{L}$ can be nondeterministically decided with certificates of $\lceil \log \alpha(n) \rceil + 1$ bits, and a set U of only three opinions. Each opinion provides an estimation of the correctness of the execution from the perspective of one process. Moreover, using arguments from combinatorial topology, we show that the result is best possible. Namely, there are distributed languages for which two opinions are insufficient, even with only three processes, and regardless of the size of the certificates.

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.

Due to space limitations, proofs and additional material can be found in a companion technical report [12].

Related work The area of *decentralized runtime verification* is concerned with a set of failure-free monitors observing the behavior of system executions with respect to some correctness property, specified in some form of temporal logic. It is known, for instance, that linear temporal logic (LTL) is not sufficient to handle all system executions, some of them requiring multi-valued logics [3]. Further references to this area appear in the recent work [22], where 3-valued semantics of LTL specifications are considered.

Deterministic distributed decision in the context of asynchronous, crash-prone distributed computing was introduced in [11] with the name *checking*, where a characterization of the tasks that are AND-checkable is provided. The results where later on extended in [14] to the set agreement task and in [13] proving nearly tight bounds on the number of opinions required to check any distributed language. In [10,15] the context of *local* distributed network computing 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.

Our construction of distributed encoding of the integers relies very much on the notion of *well-quasi-ordering* (wqo) [20]. This important tool in logic and computability has a wide variety of applications — see [21] for a survey. One important application is providing *termination arguments* in decidability results [6]. Indeed, thirteen years after publishing his undecidability result, Turing [27] 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 [8,23], as witnessed by the amount of work collected in [24]. 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 [24,26]. For more applications and related work on wqos, including rewriting systems, tree embeddings, lossy channel systems, and graph minors, see recent work [16,24].

2 Distributed Encoding of the Integers

Given a finite or infinite alphabet Σ, a *word* of size n on Σ is an ordered sequence $w = w_1, w_2, \ldots, w_n$ 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, $w' = w_{i_1}, w_{i_2}, \ldots, w_{i_k}$ with $k < n$ and $1 \leq i_1 < i_2 < \cdots < i_k \leq n$.

Definition 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, $w = \text{bin}(n), \ldots, \text{bin}(n)$. 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 $i \in \{1, \ldots, n\}$. 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:

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

Hence $A_k(0) = 2$ for every $k \geq 1$, and, for $n \geq 0$, $A_2(n) = 2^{n+2} - 2$, and $A_3(n) = 2^{2^{\cdots^2}} - 2$, where the tower is of height $n+2$. (Many versions of the Ackerman function exist, and a possible definition [25] 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:

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

Hence, α grows extremely slowly. 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 1. *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 $\lceil \log \alpha(n) \rceil + 1$ bits, where α is defined in Eq. (2).*

The proof of Theorem 1 heavily relies on the notion of *well-quasi-ordering*. Recall that a *well-quasi-ordering* (wqo) is a quasi-ordering that is well-founded and has finite antichains. 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 [23] provides an upper bound on bad sequences parametrized by a control function g and by the size $p = |\Lambda|$ of the alphabet.

Proof (Theorem 1). Consider 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. 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. For further references, we call these long bad *multi-diagonal* sequences. An example is in Fig. 2.

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 $w_i = (\mathrm{bin}(k), \mathcal{S}_i^{(n)})$ 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 $\mathcal{S}^{(n)}$, $i = 1, \ldots, n$. For each integer $n \geq 1$ and every word $w \in \Sigma^n$, we set:

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

$\mathcal{S}^{(1)} = 0$ (1st bad sequence starts)
$\mathcal{S}^{(2)} = 11$ (1st bad sequence ends)
$\mathcal{S}^{(3)} = 000$ (2nd bad sequence starts)
$\mathcal{S}^{(4)} = 0110$
$\mathcal{S}^{(5)} = 11010$
$\mathcal{S}^{(6)} = 101011$
$\mathcal{S}^{(7)} = 0101111$
$\mathcal{S}^{(8)} = 11111100$
$\mathcal{S}^{(9)} = 111110011$
$\mathcal{S}^{(10)} = 1111001111$
$\mathcal{S}^{(11)} = 11100111111$
$\mathcal{S}^{(12)} = 110011111111$
$\mathcal{S}^{(13)} = 1001111111111$
$\mathcal{S}^{(14)} = 00111111111111$
$\mathcal{S}^{(15)} = 111111111111110$
$\mathcal{S}^{(16)} = 1111111111111011$
$\mathcal{S}^{(17)} = 11111111111101111$
$\mathcal{S}^{(18)} = 111111111110111111$
$\mathcal{S}^{(29)} = 01111111111111111111111111111$
$\mathcal{S}^{(30)} = 11111111111111111111111111111$ (2nd bad sequence ends)
$\mathcal{S}^{(31)} = 0000000000000000000000000000$ (3rd bad sequence starts)
$\mathcal{S}^{(32)} = 0000000000000000000000000000110$
$\mathcal{S}^{(33)} = 0000000000000000000000000011010$
$\mathcal{S}^{(34)} = 0000000000000000000000000001101010$

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

This is a correct distributed encoding since, for every integer $n \geq 1$, there exists a word $w \in \Sigma^n$ such that $f(w) = \mathit{true}$, and, for every subword w' of w, $f(w') = \mathit{false}$. The latter holds because every subword w' must be of the form $w' = (w_{i_j})_{j=1,\ldots,m}$ 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. See [12] for a complete proof and more details on the construction of $\mathcal{S}$.

		$(x^{(i)}), \mu_i$
$M^{(1)}$	$= 0000$	$(0,0,0,0), 0$
$M^{(2)}$	$= 00110$	$(0,0,2), 0$
$M^{(3)}$	$= 011010$	$(0,2,1), 0$
$M^{(4)}$	$= 1101010$	$(2,1,1), 0$
$M^{(5)}$	$= 10101011$	$(1,1,1), 2$
$M^{(6)}$	$= 010101111$	$(0,1,1), 4$
$M^{(7)}$	$= 1111110010$	$(6,0,1), 0$
$M^{(8)}$	$= 11111001011$	$(5,0,1), 2$
$M^{(9)}$	$= 111100101111$	$(4,0,1), 4$
$M^{(10)}$	$= 1110010111111$	$(3,0,1), 6$
$M^{(11)}$	$= 11001011111111$	$(2,0,1), 8$
$M^{(12)}$	$= 100101111111111$	$(1,0,1), 10$
$M^{(13)}$	$= 0010111111111111$	$(0,0,1), 12$
$M^{(14)}$	$= 01111111111111100$	$(0,14,0), 0$
$M^{(15)}$	$= 110111111111111100$	$(2,13,0), 0$
$\vdots$	$\vdots \quad \vdots$	$\vdots$
$M^{(24)}$	$= 011111111111100111111111111$	$(0,12,0), 12$
$M^{(25)}$	$= 111111111111101111111111100$	$(14,11,0), 0$
$\vdots$	$\vdots \quad \vdots$	$\vdots$
		$(0,0,0), A_3(2) - 2$
		$(0, A_3(2)), 0$
$\vdots$	$\vdots \quad \vdots$	$\vdots$
		$(0,0), A_2(A_3(2)) - 2$
		$(A_2(A_3(2))), 0$
$\vdots$	$\vdots \quad \vdots$	$\vdots$
$M^{(F(4)-5)}$	$= 01111111111 \ldots\ldots\ldots 111111111111111111$	$(0), A_1(A_2(A_3(2))) - 2$
$M^{(F(4)-4)}$	$= 11111111111 \ldots\ldots\ldots 111111111111111111$	$(), A_1(A_2(A_3(2)))$

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

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 [10,11,13]).

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 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. 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 $val(s)$ the multiset of values in s.

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.

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 [11] 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 2. *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 Similarly to the way NP extends P, we extend the notion of distributed decision to distributed *verification*. In addition to its input x_i, process id_i receives a 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 show that it is possible to verify every distributed language using three opinions, with small size certificates. Then we show that with constant size certificates, almost constant size number of opinions are sufficient.

Verification with a constant number of 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 2. *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 of the size of the certificates.*

The proof of Theorem 2 uses arguments from combinatorial topology. Indeed, it is known (see e.g.,[17]) 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 use a similar style argument to show that *binary* consensus among three processes is not *wait-free verifiable* with only two opinions.

On the other hand, it was proved in [18] that every distributed language can be verified using only three opinions (true, false, undetermined). However, the verifier in [18] 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 3. *Every distributed language can be verified using three opinions, with certificates of size* $\lceil \log \alpha(n) \rceil + 1$ *bits for* n*-process instances.*

Verification with constant-size certificates We can reduce the size of the certificates even further, at the cost of slightly increasing the number of opinions.

Theorem 4. *Every language can be verified with 1-bit certificates, using* $2\,\alpha(n) + 1$ *opinions for* n*-dimensional instances.*

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

References

1. Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, N. Shavit: Atomic Snapshots of Shared Memory. *J. ACM* **40**(4): 873–890 (1993).
2. H. Attiya, J. Welch: Distributed Computing: Fundamentals, Simulations, and Advanced Topics. *John Wiley & Sons* (2004).
3. A. Bauer, M. Leucker, C. Schallhart: Comparing LTL Semantics for Runtime Verification. *J. Log. Comput.* **20**(3): 651–674 (2010).
4. L. Blin, P. Fraigniaud, B. 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. T. Chandra, S. Toueg: Unreliable Failure Detectors for Reliable Distributed Systems. *J. ACM* **43**(2): 225–267 (1996).
6. B. Cook, A. Podelski, A. 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. 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.
9. M. Fischer, N. Lynch, M. Paterson: Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* **32**(2): 374–382 (1985).
10. P. Fraigniaud, A. Korman, D. Peleg: Towards a Complexity Theory for Local Distributed Computing. *J. ACM* **60**(5):35 (2013).
11. P. Fraigniaud, S. Rajsbaum, C. Travers: Locality and Checkability in Wait-Free Computing. *Distributed Computing* **26**(4): 223–242 (2013).
12. P. Fraigniaud, S. Rajsbaum, C. Travers: Minimizing the Number of Opinions for Fault-Tolerant Distributed Decision Using Well-Quasi Orderings *Technical report* #hal-01237873, 2015. `https://hal.archives-ouvertes.fr/hal-01237873v1`

13. 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.
14. 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.
15. M. Göös, J. Suomela: Locally Checkable Proofs. In proc. *30th ACM Symp. on Principles of Distributed Computing* (PODC), pp. 159–168, 2011.
16. Christoph Haase, Sylvain Schmitz, Philippe Schnoebelen: The Power of Priority Channel Systems. In proc. *24th Int. Conference on Concurrency Theory* (CONCUR), Springer, LNCS 8052, pp 319–333, 2013.
17. M. Herlihy, D. Kozlov, S. Rajsbaum: Distributed Computing Through Combinatorial Topology. *Morgan Kaufmann*, 2013.
18. M. Jeanmougin: Checkability in Asynchronous Error-Prone Distributed Computing Using Few Values. *Master Thesis Report*, University Paris Diderot, 2013.
19. A. Korman, S. Kutten, D. Peleg: Proof Labeling Schemes. *Distributed Computing* **22**(4): 215–233 (2010).
20. J. Kruskal: The Theory of Well-Quasi-Ordering: A Frequently Discovered Concept. *Journal of Combinatorial Theory A* **13**(3): 297–305 (1972).
21. E. 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.
22. M. Mostafa, B. Bonakdarpour: Decentralized Runtime Verification of LTL Specifications in Distributed Systems. In proc. *IEEE Parallel and Distributed Processing Symposium* (IPDPS), pp. 494–503, 2015.
23. S. Schmitz, P. Schnoebelen: Multiply-Recursive Upper Bounds with Higman's Lemma. In proc. *38th Int'l Colloquium on Automata, Languages and Programming* (ICALP), Springer, LNCS 6756, pp. 441–452, 2011.
24. S. Schmitz, P. Schnoebelen: Algorithmic Aspects of WQO Theory. *Tech. Report* Hal#cel-00727025, 2013. `https://cel.archives-ouvertes.fr/cel-00727025v2`
25. P. Schnoebelen: Verifying Lossy Channel Systems has Nonprimitive Recursive Complexity. *Inf. Process. Lett.* **83**(5): 251–261 (2002).
26. P. Schnoebelen: Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets. In Proc. *35th Int'l Symp' on Mathematical Foundations of Computer Science* (MFCS), Springer, LNCS 6281, pp. 616–628, 2010.
27. A. Turing: Checking a Large Routine. In Report of a *Conference on High Speed Automatic Calculating Machines*, pp. 67–69, 1949.