On the Number of Opinions Needed for Fault-Tolerant Run-Time Monitoring in Distributed Systems

Pierre Fraigniaud, Sergio Rajsbaum, Corentin Travers

On the Number of Opinions Needed for Fault-Tolerant Run-Time Monitoring in Distributed Systems[*]

Pierre Fraigniaud[†]
CNRS and U. Paris Diderot, France. `pierre.fraigniaud@liafa.univ-paris-diderot.fr`

Sergio Rajsbaum[‡]
Instituto de Matemáticas, UNAM, D.F. 04510, Mexico. `rajsbaum@im.unam.mx`

Corentin Travers[§]
CNRS and U. of Bordeaux, France. `travers@labri.fr`

Abstract

Decentralized runtime monitoring involves a set of monitors observing the behavior of system executions with respect to some correctness property. It is generally assumed that, as soon as a violation of the property is revealed by any of the monitors at runtime, some recovery code can be executed for bringing the system back to a legal state. This implicitly assumes that each monitor produces a binary opinion, true or false, and that the recovery code is launched as soon as one of these opinions is equal to false. In this paper, we formally prove that, in a failure-prone asynchronous computing model, there are correctness properties for which there is no such decentralized monitoring. We show that there exist some properties which, in order to be monitored in a wait-free decentralized manner, inherently require that the monitors produce a number of opinions larger than two. More specifically, our main result is that, for every k, $1 \leq k \leq n$, there exists a property that requires at least k opinions to be monitored by n monitors. We also present a corresponding distributed monitor using at most $k+1$ opinions, showing that our lower bound is nearly tight.

1 Introduction

Runtime verification is concerned with monitoring software and hardware system executions. It is used after deployment of the system for ensuring reliability, safety, and security, and for providing fault containment and recovery. Its essential objective is to determine, at any point in time, whether the system is in a legal or illegal state, with respect to some specification. Consider a distributed system whose execution is observed by one or several monitors. Passing messages to a *central* monitor at every event leads to severe communication and computation overhead. Therefore, recent contributions [6, 9, 27] on runtime verification of distributed systems focused on

[*]Extended abstract presented at *14th International Conference on Runtime Verification (RV2014)*, September 22 - September 25, 2014 Toronto, Canada, to appear in Springer Lecture Notes in Computer Science. All authors are supported in part by the CONACYT-CNRS ECOS Nord M12M01 research grant.

[†]Additional support from the ANR project DISPLEXITY, and from the INRIA project GANG.

[‡]Additional support from UNAM-PAPIIT and LAISLA.

[§]Additional support from ANR project DISPLEXITY.

decentralized monitoring, where a set of n monitors observe the behavior of the system. As soon as a violation of the legality of the execution is revealed by any of these monitors at runtime, recovery code can be executed for bringing the system back to a legal state. For example, the recovery code can reboot the system, or release its resources. This framework implicitly assumes that each monitor i produces a binary *opinion* $o_i \in \{\text{true}, \text{false}\}$, and that the recovery code is launched as soon as one of these opinions is equal to false. In this paper, we formally prove that, in a crash-failure prone asynchronous wait-free computing model [4], there are correctness properties for which such decentralized monitoring does not exist, even if we let the number of opinions grow to an arbitrary constant $k \geq 2$.

Let us consider the following motivating example arising often in practice [8], of a system in which *requests* are sent by clients, and *acknowledged* by servers. The system is in a legal state if and only if (1) all requests have been acknowledged, and (2) every received acknowledgement corresponds to a previously sent request. Each monitor i is aware of a subset R_i of requests that has been received by the servers, and a subset A_i of acknowledgements that has been sent by the servers. To verify legality of the system, each monitor i may communicate with other monitors in order to produce some opinion o_i. In the traditional setting of decentralized monitoring mentioned in the previous paragraph, it is required that the monitors produce opinions $o_i \in \{\text{true}, \text{false}\}$ such that, whenever the system is not in a legal state, at least one monitor produces the opinion false.

In runtime monitoring, a correctness property is described by a formula in some temporal logic. In this paper, we abstract away the logic, and directly specify the property by the set of *legal* configurations of the system, that we call a *distributed language*, denoted by $\mathcal{L}$. For instance, in the request-acknowledgement example above, $\mathcal{L}$ is the set of all configurations $\{(r_i, a_i),\ i \in I\}$ such that $\cup_{i\in I}\ r_i = \cup_{i\in I}\ a_i$, where $I \subseteq [1, n]$. Indeed, this language is specifying that all observed requests have been acknowledged, and every observed acknowledgement corresponds to a previously sent request. The monitors must produce opinions enabling to distinguish the legal configurations, i.e., those in $\mathcal{L}$, from the illegal ones. In order to make up their opinions, the monitors are able to communicate among themselves, so that each monitor can potentially collect system observations of other monitors. Since we are mostly interested in lower bounds, we ask very little from the monitors, and simply require that, for any pair (C, C') of configurations with $C \in \mathcal{L}$ and $C' \notin \mathcal{L}$, the *multiset* of opinions produced by the monitors given the legal configuration C must be different from the multiset of opinions given the illegal configuration C'.

In the centralized setting, more than two logical values may be required to avoid evaluating prematurely the correctness of a property that cannot be decided solely based on a prefix of the execution, like request-acknowledgement. Hence [2, 7] extended linear temporal logic (LTL) to logics with three values (e.g., {true, false, inconclusive}). More recently, it was recognised [8] that even three values are not sufficient to monitor some properties, and thus extensions of LTL with four logical values (e.g., {true, false, probably true, probably false}) were introduced. In this paper we argue that, in an asynchronous failure-prone decentralized setting, even four values may not be sufficient.

Our results. We consider decentralized monitoring in the *wait-free* setting [4]. (See Section 2 for details about this model, and for the reasons why we chose it). Our main result is a lower bound on the number of opinions to be produced by a runtime decentralized monitor in an asynchronous system where monitors may crash. This lower bound depends solely on the language, i.e., on the correctness property being monitored. More specifically, we prove that, for any positive integer n, and for any k, $1 \le k \le n$, there exists a distributed language requiring monitors to produce at least k distinct opinions in a system with n monitors. This result holds whatever the system does with the opinions produced by the monitors. That is, our lower bound on the number of opinions is inherent to the language itself — and not to the way the opinions are handled in order to launch the recovery code to be executed in case the system misbehaves.

The number of opinions required to runtime monitor languages in a decentralized manner is actually tightly connected to an intrinsic property of each language: its *alternation number*. This parameter essentially captures the number of times a sequence of configurations of the system alternates between legal and illegal. Our main result states that, for any k, $1 \le k \le n$, there exists a language with alternation number k which requires at least k opinions to be monitored by n monitors. This bound is essentially tight, as we also design a distributed monitor which, for any k, $1 \le k \le n$, and any distributed language $\mathcal{L}$ with alternation number k, monitors $\mathcal{L}$ using at most $k + 1$ opinions in systems with n monitors.

Technically, in this paper, we establish a bridge between, on the one hand, runtime verification, and, on the other hand, distributed computability. Thanks to this bridge, we could prove our lower bound using arguments from (elementary) algebraic topology. More specifically, our impossibility result for 2 opinions is obtained using graph-connectivity techniques sharing similarities with the FLP impossibility result for consensus [15], while our general impossibility result uses higher-dimensional techniques similar to those used in set agreement impossibility results e.g. [22, 23].

As far as we know, this paper is the first one studying necessary conditions for monitoring distributed systems with failures.

Related work. The main focus in the literature is on sequential runtime verification. The monitors are event-triggered [24], where every change in the state of the system triggers the monitor for analysis. There is work also in time-triggered monitoring [10], where the monitor samples the state of the program at regular time intervals. Parallel monitoring has been addressed in [20] to some extent by focusing on low-level memory architecture to facilitate communication between application and analysis threads. The concept of separating the monitor from the monitored program is considered in, e.g., [28]. Later, [9] uses a specialized parallel architecture (GPU), to implement runtime formal verification in a parallel fashion. Efficient automatic signaling monitoring in multi-core processors is considered in [13].

Closer to our setting is decentralized monitoring. In sequential runtime verification one has to monitor the requirement based on a single behavioral trace, assumed to be collected by some global observer. A central observer basically resembles classical LTL monitoring. In contrast, in decentralized monitoring, there are several partial behavioural traces, each one collected at a component of the system. Intuitively, each trace corresponds to the view that the component has of the execution. In decentralized LTL monitoring [6] a formula ϕ is decomposed into local formulas, so monitor i evaluates locally ϕ_i, and emits a boolean-valued opinion. In our terminology, an "and interpretation" is used. That is, it is assumed a global violation can always be detected locally by a process. In addition, it is assumed the set of local monitors communicate over a synchronous bus with a global clock. The goal is to keep the communication among monitors minimal. In [26] the focus is in monitoring safety properties of a distributed program's execution, also using an "and interpretation". The decentralized monitoring algorithm is based on formulae written in a variant of past time LTL. For the specific case of relaxed memory models, [11] presents a technique for monitoring that a program has no executions violating sequential consistency. There is also work [19] that targets physically distributed systems, but does not focus on distributed monitoring.

To the best of our knowledge, the effects of asynchrony and failures in a decentralized monitoring setting were considered for the first time in [17]. We extend this previous work in two ways. First, we remove the restriction that the monitors can produce only two opinions. Second, [17] investigated applications to locality, while here we extend the framework and adapt it to be able to apply it to a more general decentralized monitoring setting.

Related work in the distributed computing literature includes seminal papers such as [12] for stable property detection in a failure-free message-passing environment, and [5] for distributed program checking in the context of self-stabilization.

Organization of this paper. The distributed system model is in Section 2. Distributed languages and wait-free monitoring are presented in Section 3. In Section 4 we present the example of monitoring leader election. Our main result is in Section 5. Its proof is presented in Section 6. We conclude the paper and mention some open problems in Section 7.

2 Distributed system model

There are many possible computation and communication models for distributed computation. Here we assume wait-free asynchronous processes that may fail by crashing, communicating by reading and writing a shared memory. This model serves as a good basis to study distributed computability: results in this model can often be extended to other popular models, such as when up to a fixed number of processes can crash (in a dependent or independent way). Also, message-passing, or various networking models that limit direct process-to-process connectivity, are essentially computationally equivalent or less powerful than shared memory. We recall here the main features of the wait-free model, and refer to textbooks such as [4] for a more detailed description, as well as for the relation to other distributed computing models.

The asynchronous read/write shared memory model assumes a system consisting of n asynchronous processes. Let $[n] = \{1, \dots, n\}$. We associate each process to an integer in $[n]$. 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 *wait-free* distributed algorithms, in which a process never "waits" for another process to produce some intermediate result. This is because any number of processes may crash (and thus the expected result may never be produced).

The processes communicate through a shared memory composed of atomic registers, organised as an array of n single-writer/multiple-reader (SWMR) registers, one per process. Register $i \in [n]$ supports the operation $read()$ that returns the value stored in the register, and can be executed by any process. It also support de operation $write(v)$ that writes the value v in the register, and can be executed only by process i.

In our algorithms we use a *snapshot* operation by which a process can read all n SWMR 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 snapshot operation (snapshots are linearizable). Snapshots can be implemented by a wait-free algorithm (any number of processes may crash) using only the array of n SWMR registers [1] (see also textbooks such as [25]). Thus, we may assume snapshots are available to the processes, without loss of generality. The algorithms are simplified, as well as the proofs of our theorems, without modifying the outcomes of our results.

In a *distributed algorithm* each process starts with an *input value*, repeats a loop N times, consisting of writing to its register, taking a snapshot and making local computations[1]. At the end each process produces an *output value.* In a *step*, a process performs an operation on the registers (i.e., writes or snapshots). A *configuration* completely describes the state of the system. That is, a configuration specifies the state of each register as well as the local state of each process. An *execution* is a (finite) sequence of alternating configurations and steps, starting and ending in a configuration. A process *participates* in an execution if it takes at least one step in the execution. We assume that the first step of a process is a write, and it writes its input.

3 Distributed languages and wait-free monitoring

3.1 Distributed languages

Let A be an alphabet of symbols, representing the set of possible values produced by some distributed algorithm to be monitored. Each process $i \in [n]$ has a read-only variable, $input_i$, initially equal to a symbol $\perp$ (not in A), and where the value to be monitored is deposited. We consider only the simplest scenario, where these variables change only once, from the value $\perp$, to a value in A. The goal is for the processes to monitor that, collectively, the values deposited in these variables are correct.

Formally, consider an execution $C_0, s_1, C_1, \ldots$, where each C_i is a configuration and each s_i is a step (write or snapshot) by some process, and C_0 is the initial configuration where all SWMR registers are empty. We assume the first step by a process i is to write its input, and is taken only once its variable $input_i$ is initialized to a value in A. Thus, s_1 is a write step by some process.

The correctness specification to be monitored is usually stated as a global predicate in some logic (e.g. [13, 14]). We rephrase the predicate in terms of what we call a *distributed language.* An *instance* over alphabet A (we may omit A when clear from the context) is a set of pairs $s = \{(\mathrm{id}_1, a_1), \ldots, (\mathrm{id}_k, a_k)\}$, where $\{\mathrm{id}_1, \ldots, \mathrm{id}_k\} \subseteq [n]$ are distinct process identities, and $a_1, \ldots, a_k$ are (not necessarily distinct) elements of A. A distributed language $\mathcal{L}$ over the alphabet A is a collection of instances over A. Given a language $\mathcal{L}$, we say that an instance s is *legal* if $s \in \mathcal{L}$ and *illegal* otherwise.

Let $s = \{(\mathrm{id}_1, a_1), \ldots, (\mathrm{id}_k, a_k)\}$ be an instance over A. We denote by $\mathrm{ID}(s)$ the set of identities in s, $\mathrm{ID}(s) = \{\mathrm{id}_1, \ldots, \mathrm{id}_k\}$. The multiset of values in s is denoted by $\mathit{val}(s)$ (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).

Note that an instance s can describe an assignment of values from A to the input variables of a subset of processes. More precisely, consider an execution $C_0, s_1, C_1, \ldots, s_k, C_k$, $k \geq 1$. Suppose the processes that have taken steps in this execution are those in P, $P \subseteq [n]$. This execution defines the instance $s = \{(\mathrm{id}_1, a_1), \ldots, (\mathrm{id}_k, a_k)\}$ over A, where $\mathrm{ID}(s) = P$ and a_i is the first value written by process id_i. A configuration C_k also defines an instance, given by the input variables of processes that have written at least once (from the local state of a process, one can deduce if it has already executed a write operation).

An execution is *correct* if and only if its instance s is in $\mathcal{L}$. If the execution is correct, then processes in $\mathrm{ID}(s)$ have values as specified by the language (and the other processes have not yet

[1]If the set of possible input values is finite, all processes may execute the loop the same number of times, N (e.g. see [3]).

been assigned a value or may be slow in announcing their values).

Consider for example the language `req-ack`, which captures a simplified version of the request-acknowledgment problem mentioned in the introduction, in which no more than q requests are sent by the clients. Requests and acknowledgments are identified with integers in $[q]$. A process id_i may know of some subset of requests $r_i \subseteq [q]$, and some subset of acknowledgments $a_i \subseteq [q]$. The language `req-ack` over alphabet $A = 2^{[q]} \times 2^{[q]}$ is defined by instances s as follows

$$s = \Big\{(\mathrm{id}_1, (r_1, a_1)), \ldots, (\mathrm{id}_k, (r_k, a_k))\Big\} \in \texttt{req-ack} \iff \bigcup_{1\le i\le k} r_i = \bigcup_{1\le i\le k} a_i.$$

For each process i, the sets r_i and a_i denote the (possibly empty) sets of requests and acknowledgments, respectively, that process i is aware of. An instance is legal if and only if every request has been acknowledged.

As another example, consider *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:

$$s = \{(\mathrm{id}_1, \ell_1), \ldots, (\mathrm{id}_k, \ell_k)\} \in \texttt{leader} \iff \exists i \in [k] : \mathrm{id}_i = \ell_1 = \cdots = \ell_k. \tag{1}$$

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

3.2 Decentralized monitoring

Monitoring the correctness specified by a language $\mathcal{L}$ involves two components: an *opinion-maker* M, and an *interpretation* μ. The opinion-maker is a distributed algorithm executed by the processes enabling each of them to produce an individual *opinion* about the validity of the outputs of the system. We call the processes running this algorithm *monitors*, and the (finite) set of possible individual opinions U, the *opinion set*.

The interpretation μ specifies the way one should interpret the collection of individual opinions produced by the monitors about the validity of the monitored system. We use the minimal requirement that the opinions of the monitors 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 processes 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 [17, 16] is as follows. For every multi-set of opinions S, we 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 *monitor for* $\mathcal{L}$ is a pair (μ, M), as follows.

- The opinion-maker M is a distributed wait-free algorithm that outputs an opinion u_i at every process i. The input of process i is any element a_i of A (assigned to its read-only variable $input_i$). Each process i is required to produce an opinion u_i such that: (1) every non-faulty process eventually produces an output (termination), and (2) if process i outputs u_i, then we must have: $u_i \in U$ (validity).

- Consider any execution of M where all participating processes have decided an opinion. If the instance s corresponding to the execution is legal, i.e., $s \in \mathcal{L}$, the monitors must produce a multiset of opinions $S \in \mathbf{Y}$, and if the instance s is illegal, i.e., $s \notin \mathcal{L}$, then they must produce a multiset of opinions in $\mathbf{N}$.

The paper focusses on the following question: given a distributed language $\mathcal{L}$, how many opinions are needed to monitor $\mathcal{L}$?

3.3 Opinion and alternation numbers

As stated above, we are interested in the smallest size $|U|$ of the opinion set enabling the monitors, after the execution of some distributed algorithm, to output opinions that distinguish legal instances from illegal ones. Hence, we focus on the following parameter associated with every distributed language.

Definition 1 (Opinion number) *Let $\mathcal{L}$ be a distributed language on n processes. The* opinion number *of $\mathcal{L}$ is the smallest integer k for which there exists a monitor (μ, M) for $\mathcal{L}$ using a set of at most k opinions. It is denoted by* $\#\text{opinion}(\mathcal{L})$.

As we shall see, there are monitors using a small number of opinions, independent of the size of the alphabet used to define $\mathcal{L}$, and depending only on the number n of processes. The opinion number is shown to be related to a combinatorial property of languages, captured by the notion of *alternation number*. Given a language $\mathcal{L}$ over the alphabet A, the alternation number of $\mathcal{L}$ is the length of a longest increasing sequence of instances $s_1, \ldots, s_k$ with alternating legality. More formally:

Definition 2 (Alternation number) *Let $\mathcal{L}$ be a distributed language. The* alternation number *of $\mathcal{L}$ is the largest integer k for which there exists instances $s_1, \ldots, s_k$ such that, for every i, $1 \leq i < k$, $s_i \subset s_{i+1}$, and either $(s_i \in \mathcal{L}) \wedge (s_{i+1} \notin \mathcal{L})$ or $(s_i \notin \mathcal{L}) \wedge (s_{i+1} \in \mathcal{L})$. It is denoted by* $\#\text{altern}(\mathcal{L})$.

Clearly, the alternation number is at most n since an instance has at most n elements.

4 Monitoring leader election

As a warm up example, let us show that the language `leader` of Equation 1 can be monitored using three opinions, namely, that $\#\text{opinion}(\texttt{leader}) \leq 3$. To establish this result, we describe a monitor for `leader`, called *traffic-light*. The set of opinions consists of three values, namely $\{\text{red}, \text{orange}, \text{green}\}$. Recall that the input of each process $i \in [n]$ is a value ℓ_i where $\ell_i \in [n]$ is supposed to be the identity of the leader. The opinion maker works as follows. Each monitor i writes its identity and it own input ℓ_i in shared memory, and then reads the whole memory with a snapshot operation. The snapshot returns a set of pairs, $s_i = \{(\text{id}_j, \ell_j), j \in I\}$ for some I, that includes the values written so far in the memory. Recall that processes run asynchronously, hence a process may collect values from only a subset of all processes. Process i decides "green" if every process in s_i agrees on the same leader, and the ID of the common leader is the ID of one of the processes in s_i. Instead, if two or more processes in s_i have distinct leaders, then process i decides "red". In the somewhat "middle" case in which every process in s_i agrees on the same leader (i.e., same ID), but the ID of the common leader is not an ID of a process in s_i, then process i decides "orange".

More formally, the traffic-light opinion maker uses two procedures: "agree" and "valid". Given a set $s = \{(\mathrm{id}_1, \ell_1),\ldots,(\mathrm{id}_k, \ell_k)\}$ of pairs $(\mathrm{id}_i, \ell_i) \in [n] \times [n]$, agree$(s)$ is true if and only if $\ell_i = \ell_j$ for every i, j, $1 \leq i, j \leq k$. For a same s, valid(s) is true if and only if, for every ℓ_i, $1 \leq i \leq k$, there exists $j, 1 \leq j \leq k$ such that $\mathrm{id}_j = \ell_i$. Each process performs the pseudo-code below:

```
Opinion-maker at process $p$ with input $\ell$:
write $(\mathrm{ID}(p), \ell)$ to $p$'s register ;
snapshot memory, to get $s = \{(\mathrm{id}_1, \ell_1), \ldots, (\mathrm{id}_k, \ell_k)\}$;
if agree($s$) and valid($s$) then decide "green"
else if agree($s$) but not valid($s$) then decide "orange" else decide "red".
```

The interpretation of the opinions produced by the monitors is the following. An opinion u_i produced by process i is an element of the set $U = \{\text{green, orange, red}\}$. The opinion-maker produces a multi-set u of opinions. We define the yes-set $\mathbf{Y}$ as the set of all multi-sets u with no red elements, and at least one green element. Hence, $\mathbf{N}$ is composed of all multi-sets u with at least one red element, or with no green elements.

Now, one can easily check (see Appendix A.1) that the traffic-light monitor satisfies the desired property. That is, for every set $s = \{(\mathrm{id}_1, \ell_1), \ldots, (\mathrm{id}_k, \ell_k)\}$ of pairs $(\mathrm{id}_i, \ell_i) \in [n] \times [n]$, if u denotes the multi-set of opinions produced by the monitors, then we have

$$s \in \texttt{leader} \iff u \in \mathbf{Y}.$$

Interestingly enough, one can prove that the language `leader` *cannot be monitored using fewer than three opinions.* Namely,

Proposition 1 *#opinion(`leader`) = 3.*

Crucially, the fact that three opinions are required, and that, in particular, the opinions true and false are not sufficient, is an inherent property of the language `leader`, independently of the opinion-maker algorithm, and independently of the interpretation of the opinions produced by the monitors. The lower bound argument enabling to establish this result is not hard (see Appendix A.1), but uses a fundamental theorem about two-process read/write wait-free computation: the graph of executions is connected (e.g. see [3]).

As we mentioned before, the number of opinions required to monitor a distributed language is strongly related to its alternation number. The sequence of instances

$$s_1 = \{(1, 2)\},\ s_2 = \{(1, 2), (2, 2)\}, \text{ and } s_3 = \{(1, 2), (2, 2), (3, 3)\}$$

satisfies $s_1 \subset s_2 \subset s_3$. Moreover s_1 and s_3 are illegal, while s_2 is legal (as far as `leader` is concerned). We thus infer that the alternation number of `leader` is at least 3. In fact, it can be shown that its alternation number is exactly 3. (see Appendix A.2). Namely,

Proposition 2 *#altern(`leader`) = 3.*

Intuitively, the alternation between legal and illegal instances forces the processes to use three opinions. Given s_1, process 1 may say that the instance is "probably illegal" (orange), while, given s_2, process 2 may say that the instance is "potentially legal" (green). Only process 3, given s_3, can declare that the instance is "definitively illegal" (red), no matter the number of further processes that may show up.

5 Main result

In this section, we state our main result, that is, a lower bound on the number of opinions needed to monitor languages with n monitors.

Theorem 1 *For any $n \geq 1$, and every k, $1 \leq k < n$, there exists a language $\mathcal{L}$ on n processes, with alternation number k, that requires at least k opinions to be monitored. For $k = n$, there exists a language $\mathcal{L}$ on n processes, with alternation number n, that requires at least $n + 1$ opinions to be monitored.*

In other words, there are system properties which require a large number of opinions to be monitored. Before dwelling into the details of the proof of Theorem 1, we want to stress the fact that our lower bound is essentially the best that can be achieved in term of alternation number. Indeed, Theorem 1 says that, for every k, there exists a language $\mathcal{L}$ with alternation number k such that $\#\text{opinion}(\mathcal{L}) \geq \#\text{altern}(\mathcal{L})$. We show that this lower bound is essentially tight. Indeed, we establish the existence of a *universal* monitor that can monitor all distributed languages using a number of opinions equal roughly to the alternation number. More specifically, we show that, for every k, and for every language $\mathcal{L}$ with opinion number k, we have $\#\text{opinion}(\mathcal{L}) \leq \#\text{altern}(\mathcal{L}) + 1$.

Theorem 2 *There exists a monitor which, for every $k \geq 1$, monitors every language with alternation number k using at most $k + 1$ opinions.*

Since the alternation number of a language on n processes is at most n, Theorem 2 yields the following.

Corollary 1 *There exists a monitor which, for every $n \geq 1$, monitors every language on n processes, using at most $n+1$ opinions. Moreover, this monitor uses at most $k+1$ opinions for every execution in which at most k processes participate.*

A direct proof of Corollary 1 can be found in Appendix B. The interest of this direct proof is that it displays the flavor of some of the technical ingredients used for establishing Theorem 2. The full proof of Theorem 2 can be found in Appendix C.

It is worth noticing that the monitor of Corollary 1 has an interpretation μ which does not depend at all on the language to be monitored, not even on the number of processes involved in the language. (The same holds for Theorem 2). The opinion-maker (as well as the one for Theorem 2), does however depend on the language, but only up to a limited extent. Indeed, the general structure of the opinion-maker is independent of the language. It simply uses a black box that returns whether $s \in \mathcal{L}$ for any instance s. Apart from this, the opinion-maker is essentially independent of the language. In this sense it is *universal.*

The rest of the paper is dedicated to describing the main ideas of the proof of our main result.

6 Orientation-detection tasks, and proof of Theorem 1

To establish our lower bound, we show that the design of distributed runtime monitors using few opinions is essentially equivalent to solving a specific type of tasks, that we call *orientation-detection* tasks. This equivalence is made explicit thanks to an *equivalence lemma* (Lemma 1). Introducing orientation-detection tasks requires elementary notions of combinatorial topology.

6.1 Tasks and combinatorial topology terminology

When solving a distributed task[2], each process starts with a private input value and has to eventually decide irrevocably on an output value. (In our setting, the input value of a process is a symbol in a given alphabet A, and the output value is an opinion). A process $i \in [n]$ is initially not aware of the inputs of other processes. Consider an execution where only a subset of k processes participate, $1 \leq k \leq n$. These processes have distinct identities $\{\mathrm{id}_1, \ldots, \mathrm{id}_k\}$, where for every $i \in [k]$, $\mathrm{id}_i \in [n]$. A set $s = \{(\mathrm{id}_1, x_1), \ldots, (\mathrm{id}_k, x_k)\}$ is used to denote the input values, or output values, in the execution, where x_i denotes the value of the process with identity id_i — either an input value (e.g., a symbol in a given alphabet A), or a output value (e.g., an opinion).

The monitor task. An opinion-maker M for a language $\mathcal{L}$ on n processes with opinion set U, and interpretation $\mu = (\mathbf{Y}, \mathbf{N})$ is a distributed wait-free algorithm that solves the following *monitor task*. Any instance over alphabet A is a possible assignment of inputs in A to the processes. If process i has input $a_i \in A$, then i is required to produce as output an opinion $u_i \in U$ such that, in addition to satisfying termination and validity, it also satisfy *consistency*, defined as follows. Consider any execution, where I is the set of processes that do not crash, and all others crash without taking any steps. Let $s = \{(\mathrm{id}_i, a_i), i \in I\}$, and let $u = \{u_i, i \in I\}$ denote the multiset of opinions that are eventually output by the processes in I. We must have: $s \in \mathcal{L} \iff u \in \mathbf{Y}$.

Simplices and complexes. Let s' be a subset of a "full" set $s = \{(1, x_1), \ldots, (n, x_n)\}$, i.e., a set s such that $\mathrm{ID}(s) = [n]$. Since any number of processes can crash, all such subsets s' are of interest for taking into account executions where only processes in $\mathrm{ID}(s')$ participate. Therefore, the set of possible input sets forms a *complex* because its sets are closed under containment. Similarly, the set of possible output sets also form a complex. Following the standard terminology of combinatorial topology, the sets of a complex are called *simplexes*. Hence every set s' as above is a simplex.

More formally, 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. The *dimension* of a simplex s is $|s| - 1$, the dimension of $\mathcal{K}$ is the largest dimension of its simplexes, and $\mathcal{K}$ is *pure* of dimension k if every simplex belongs to a k-dimensional simplex. In distributed computing, the simplexes (and complexes) are often *chromatic*, since each vertex v of a simplex is labeled with a distinct process identity $i \in [n]$.

A *distributed task* T is formally described by a triple $(\mathcal{I}, \mathcal{O}, \Delta)$ where $\mathcal{I}$ and $\mathcal{O}$ are pure $(n-1)$-dimensional complexes, and Δ is a map from $\mathcal{I}$ to the set of non-empty sub-complexes of $\mathcal{O}$, satisfying $\mathrm{ID}(t) \subseteq \mathrm{ID}(s)$ for every $t \in \Delta(s)$. We call $\mathcal{I}$ the input complex, and $\mathcal{O}$ the output complex. Intuitively, Δ specifies, for every simplex $s \in \mathcal{I}$, the valid outputs $\Delta(s)$ for the processes in $\mathrm{ID}(s)$ that may participate in the computation. We assume that Δ is (sequentially) computable.

Given any finite set U and any integer $n \geq 1$, we denote by $\mathit{complex}(U, n)$ the $(n-1)$-dimensional *pseudosphere* [22] complex induced by U: for each $i \in [n]$ and each $x \in U$, there is a vertex labeled (i, x) in the vertex set of $\mathit{complex}(U, n)$. Moreover, $u = \{(\mathrm{id}_1, u_1), \ldots, (\mathrm{id}_k, u_k)\}$ is a simplex of $\mathit{complex}(U, n)$ if and only if u is properly colored with identities, that is $\mathrm{id}_i \neq \mathrm{id}_j$ for every $1 \leq i < j \leq k$. In particular, $\mathit{complex}(\{0, 1\}, n)$ is (topologically equivalent) to the $(n-1)$-dimensional sphere. For $u \in \mathit{complex}(U, n)$, we denote by $\mathit{val}(u)$ the multiset formed of all the

[2]A task is the basic distributed computing problem, defined by a set of inputs to the processes and for each input to the processes, a set of legal outputs of the processes – see, e.g., [22].

values in U corresponding to the processes in u.

6.2 Orientation-detection tasks

An *oriented complex*[3] is a complex whose every simplex s is assigned a sign, $sign(s) \in \{-1,+1\}$. Given an oriented input complex, $\mathcal{J}$, a natural task consists in computing distributively the sign of the actual input simplex. That is, each process is assigned as input a vertex of $V(\mathcal{J})$, and the set of all the vertices assigned to the processes forms a simplex $s \in \mathcal{J}$. Ideally, one would like that processes individually decide "yes" if the simplex is oriented $+1$ and "no" otherwise. However, this is impossible in general because processes do not have the same view of the execution, and any form of non-trivial agreement cannot be solved in a wait-free manner [15]. Thus, we allow processes to express their knowledge through values in some larger set U.

Definition 3 (Orientation detection task) *Let $\mathcal{J}$ be a $(n-1)$-dimensional oriented complex. A task $T = (\mathcal{J},\mathcal{U},\Delta)$, with $\mathcal{U} = complex(U,n)$ for some set U, is an* orientation detection *task for $\mathcal{J}$ if and only if for every two $s, s' \in \mathcal{J}$, and every $t \in \Delta(s)$ and $t' \in \Delta(s')$: $sign(s) \neq sign(s') \Rightarrow val(t) \neq val(t')$.*

Hence, to detect the orientation of a simplex s, the processes i, $i \in I \subseteq [n]$, occurring in a simplex s have to collectively decide a multiset $val(t) = \{val(i), i \in I\}$ of values in U, where $val(i)$ denotes the value decided by process i. If $\mathcal{J}$ is non-trivially oriented, i.e., if there exist $s, s' \in \mathcal{J}$ of the same dimension, with $sign(s) \neq sign(s')$, then no orientation-detection tasks for $\mathcal{J}$ exists with $|U| = 1$, because one must be able to discriminate the different orientations of s and s'. Instead, for every oriented complex $\mathcal{J}$, there exists an orientation-detection task for $\mathcal{J}$ with $|U| = 2$. To see why, consider the task $T = (\mathcal{J},\mathcal{U},\Delta)$, where $\mathcal{U}$ is the $(n-1)$-dimensional sphere, and Δ maps every k-dimensional simplex $s \in \mathcal{J}$ with $sign(s) = -1$ (resp., $+1$) to the k-dimensional simplex $t \in \mathcal{U}$ with $val(t) = \{0, 0, \ldots, 0\}$ (resp., $val(t) = \{1, 1, \ldots, 1\}$). However, this latter task is not necessarily wait-free solvable (i.e., solvable in our context of asynchronous distributed computing where any number of processes can crash). The complexity of detecting the orientation of an oriented complex $\mathcal{J}$ is measured by the smallest k for which there exists an orientation-detection task $T = (\mathcal{J},\mathcal{U},\Delta)$ that is wait-free solvable, with $\mathcal{U} = complex(U,n)$, and $|U| = k$.

In the next subsection, we show that the problem of finding the minimum-size set U for detecting the orientation of an arbitrary given oriented complex $\mathcal{J}$ is essentially equivalent to finding the minimum-size set of opinions U for monitoring a language $\mathcal{L}_{\mathcal{J}}$ induced by $\mathcal{J}$ (and its orientation).

6.3 Equivalence lemma

This section shows that the notion of monitoring and the notion of orientation-detection are essentially two sides of the same coin.

Let $\mathcal{L}$ be a n-process distributed language defined over an alphabet A. We define $\mathcal{J}_{\mathcal{L}} = complex(n, A)$. That is, for every collection $\{a_1, \ldots, a_k\}$ of at most k elements of A, $1 \leq k \leq n$, and every k-subset $\{\mathrm{id}_1, \ldots, \mathrm{id}_k\} \subseteq [n]$ of distinct identities, $\{(\mathrm{id}_1, a_1), \ldots, (\mathrm{id}_k, a_k)\}$ is a simplex in $\mathcal{J}_{\mathcal{L}}$. Let us orient $\mathcal{J}_{\mathcal{L}}$ as follows. For every simplex $s \in \mathcal{J}_{\mathcal{L}}$, we define:

$$sign(s) = \begin{cases} +1 & \text{if } s \in \mathcal{L}; \\ -1 & \text{otherwise.} \end{cases}$$

[3]In the case of chromatic manifolds, our definition is equivalent to usual definition of orientation in topology textbooks.

Conversely, let $\mathcal{J}$ be a well-formed oriented complex. We say that an oriented complex $\mathcal{J}$ on n processes is *well-formed* if for every $I \subseteq [n]$, there exists $s, s' \in \mathcal{J}$ with $\mathrm{ID}(s) = \mathrm{ID}(s') = I$ and $sign(s) = -sign(s')$. We set $\mathcal{L}_{\mathcal{J}}$ as the n-process language defined over the alphabet $A = \{+1, -1\} \times V(\mathcal{J})$. That is, each element of A is a pair (σ, v) where σ is a sign in $\{+1, -1\}$ and v a vertex of $\mathcal{J}$. The language $\mathcal{L}_{\mathcal{J}}$ is the set of instances $s = \{(\mathrm{id}_1, (\sigma_1, v_1)), \dots, (\mathrm{id}_k, (\sigma_k, v_k))\}$ specified as follows:

$$s \in \mathcal{L}_{\mathcal{J}} \iff \begin{cases} t = \{(\mathrm{id}_1, v_1), \dots, (\mathrm{id}_k, v_k)\} \text{ is a simplex of } \mathcal{J}, \\ \text{and } sign(t) = \sigma_i \text{ for every } i,\ 1 \le i \le k. \end{cases}$$

That is, in a legal instance, each process is assigned a vertex of some simplex $t \in \mathcal{J}$ together with the orientation of t.

We have now all ingredients to state formally the first main ingredient toward establishing Theorem 1: the equivalence between language-monitoring and orientation-detection.

Lemma 1 (Equivalence lemma)
• *Let $\mathcal{L}$ be a n-process language. If there exists $k \ge 1$ and a wait-free solvable orientation-detection task for $\mathcal{J}_{\mathcal{L}}$ using values in some set of size k, then there exists a monitor for $\mathcal{L}$ using at most k opinions.*
• *Let $\mathcal{J}$ be a well-formed oriented complex, and let $k \ge 1$. If no orientation-detection task for $\mathcal{J}$ is wait-free solvable using k values, then the language $\mathcal{L}_{\mathcal{J}}$ requires at least $k+1$ opinions to be monitored.*

The proof of Lemma 1 can be found in Appendix D. This lemma establishes an equivalence between wait-free solving orientation-detection tasks and monitoring a language with few opinions. It can be shown that, in addition, this lemma preserves alternation numbers in the following sense. The concept of alternation number (for languages) can be similarly defined for oriented complexes: for an oriented complex $\mathcal{J}$, the alternation number of $\mathcal{J}$ is the length of a longest increasing sequence of simplexes of $\mathcal{J}$ with alternating orientations. Formally:

Definition 4 (Alternation number of oriented complexes) *Let $\mathcal{J}$ be an oriented complex. The* alternation number*, $\#\mathrm{altern}(\mathcal{J})$, of $\mathcal{J}$ is the largest integer k for which there exists $s_1, \dots, s_k \in \mathcal{J}$ such that, for every i, $1 \le i < k$, $s_i \subset s_{i+1}$ and $sign(s_i) \ne sign(s_{i+1})$.*

The equivalence established in Lemma 1 preserves alternation number as stated by the following result. (see proof in Appendix E):

Lemma 2 *For every language $\mathcal{L}$, and every well-formed oriented complex $\mathcal{J}$, we have $\#\mathrm{altern}(\mathcal{J}_{\mathcal{L}}) = \#\mathrm{altern}(\mathcal{L})$ and $\#\mathrm{altern}(\mathcal{L}_{\mathcal{J}}) \le \#\mathrm{altern}(\mathcal{J}) + 1$.*

6.4 Sketch of the proof of Theorem 1

We only sketch the proof of Theorem 1. (see full proof in Appendix F). We use the correspondence between monitors and orientation-detection tasks as stated in Lemma 1, and focus on orientation-detection tasks. Given $k, 1 \le k < n$, we carefully build an oriented complex $\mathcal{J}$ with alternation number $k-1$ and shows that any orientation-detection task with input complex $\mathcal{J}$ cannot be solved wait-free with $k-1$ values or less. Therefore, by the equivalence Lemma (Lemma 1), the language

$\mathcal{L}_{\mathcal{J}}$ induced by $\mathcal{J}$ requires at least k values to be monitored. To complete the proof, we establish that the alternation number of $\mathcal{L}_J$ satisfies $\#\text{altern}(\mathcal{L}_J) = \#\text{altern}(\mathcal{J}) + 1 = k$. (The case $k = n$ is similar, except that we construct $\mathcal{J}$ with alternation number n, and $\#\text{altern}(\mathcal{L}_{\mathcal{J}}) = \#\text{altern}(\mathcal{J})$.).

The main challenge lies in constructing, and orienting the complex $\mathcal{J}$ in such a way that no orientation-detection task with input $\mathcal{J}$ is wait-free solvable with less than k values. One important ingredient in the proof is an adaptation of Sperner's Lemma to our setting. To get an idea of how the proof proceeds, consider a ℓ-dimensional simplex $s \in \mathcal{J}$ whose all $(\ell - 1)$-dimensional simplexes have sign -1, but one which has sign $+1$. (see Figure 1).

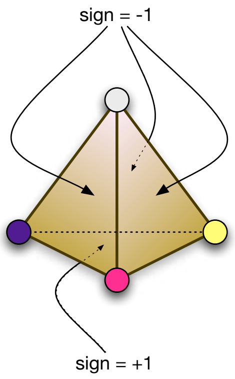

Figure 1: Using Sperner's Lemma

Assume moreover that ℓ values only are used to encode the signs of these faces. Recall that any wait-free distributed algorithm induces a mapping from a subdivision of the input complex to the output complex [23]. By Sperner's Lemma, we prove that, whatever the opinion-maker does, at least one ℓ-dimensional simplex s' resulting from the subdivision of s satisfies $|val(s')| = \ell + 1$. That is, $\ell + 1$ values are used to determine the orientation of s, for every monitor (μ, M). Appendix F addresses these technical difficulties with care. $\square$

7 Conclusions and future work

We investigated the minimum number of opinions needed for runtime monitoring in an asynchronous distributed system where any number of processes may crash. We considered the simplest case, where each process outputs a single value just once, and the monitors verify that the values collectively satisfy a given correctness condition. A correctness condition is specified by a collection of legal sets of these values, that may occur in an execution. Each monitor expresses its opinion about the correctness of the set of outputs, based on its local perspective of the execution. We proved lower bounds on the number of opinions, and presented distributed monitors with nearly the same number of opinions.

Many avenues remain open for future research. It would be interesting to derive a temporal logic framework that corresponds to ours, and that associates to opinions a formal meaning in the logic. In our setting the processes produce just one output and the monitors must verify that, collectively, the set of outputs produced is correct. It would of course be interesting to extend our results to the case where each process produces a sequence of output values. Also, opinions are anonymous.

The interpretation specifies which multisets of opinions indicate a violation, independently of the identities of the monitors that output them. We do not know whether or not taking into account the identities would help reducing the total number of opinions needed. Finally, it would be interesting to extend our results to other models, such as t-resilient models in which not more than t processes may fail.

References

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, 1993.

[2] O. Arafat, A. Bauer, M. Leucker, and C. Schallhart. Runtime verification revisited. Technical Report TUM-I0518, Technischen Universität München, 2005.

[3] H. Attiya and S. Rajsbaum. The Combinatorial Structure of Wait-Free Solvable Tasks. *SIAM J. Comput.*, 31(4):1286–1313, 2002.

[4] H. Attiya and J. L. Welch. *Distributed computing: fundamentals, simulations and advanced topics.* Wiley, USA, 2004.

[5] B. Awerbuch and G. Varghese. Distributed Program Checking: A Paradigm for Building Self-stabilizing Distributed Protocols (Extended Abstract). *SFCS*, pp. 258–267. IEEE, 1991.

[6] A. Bauer and Y. Falcone. Decentralised LTL monitoring. *Formal Methods*, lncs #7436, pp. 85–100. Springer, 2012.

[7] A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. *FSTTCS*, lncs #4337, pp. 260–272. Springer, 2006.

[8] A. Bauer, M. Leucker, and C. Schallhart. Comparing LTL semantics for runtime verification. *J. Log. and Comput.*, 20(3):651–674, 2010.

[9] S. Berkovich, B. Bonakdarpour, and S. Fischmeister. Gpu-based runtime verification. *IPDPS*, pp. 1025–1036. IEEE, 2013.

[10] B. Bonakdarpour, S. Navabpour, and S. Fischmeister. Sampling-based runtime verification. *Formal Methods*, pp. 88–102. Springer, 2011.

[11] J. Burnim, K. Sen, C. Stergiou. Sound and complete monitoring of sequential consistency for relaxed memory models. *TACAS*, lncs#6605, pp. 11–25, 2011.

[12] K. M. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Trans. Comput. Syst.*, 3(1):63–75. ACM, 1985.

[13] H. Chauhan, V. K. Garg, A. Natarajan, and N. Mittal. A distributed abstraction algorithm for online predicate detection. *SRDS*, pp. 101–110. IEEE, 2013.

[14] R. Cooper and K. Marzullo. Consistent detection of global predicates. *Workshop on Parallel and Distributed Debugging*, pp. 167–174. ACM Press, 1991.

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

[16] P. Fraigniaud, A. Korman, and D. Peleg. Local distributed decision. *FOCS*, pp. 708–717. IEEE, 2011.

[17] P. Fraigniaud, S. Rajsbaum, and C. Travers. Locality and checkability in wait-free computing. *Distributed Computing*, 26(4):223–242, 2013.

[18] P. Fraigniaud, S. Rajsbaum, and C. Travers. On the Number of Opinions Needed for Fault-Tolerant Run-Time Monitoring in Distributed Systems *Rapport de recherche*, hal-01011079, 2014.

[19] A. Genon, T. Massart, C. Meuter. Monitoring distributed controllers: When an efficient LTL algorithm on sequences is needed to model-check traces. *Formal Methods* , lncs #4085, pp. 557–57, 2006.

[20] J. Ha, M. Arnold, S. M. Blackburn, and K. S. McKinley. A concurrent dynamic analysis framework for multicore hardware. *OOPSLA*, pp. 155–174. ACM, 2009.

[21] M. Henle. *A Combinatorial Introduction to Topology.* Dover, 1983.

[22] M. Herlihy, D. Kozlov, and S. Rajsbaum. *Distributed Computing Through Combinatorial Topology.* Morgan Kaufmann-Elsevier, 2013.

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

[24] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Form. Methods Syst. Des.*, 19(3):291–314, 2001.

[25] M. Raynal. *Concurrent Programming - Algorithms, Principles, and Foundations.* Springer, 2013.

[26] K. Sen, A. Vardhan, G. Agha, and G. Rosu. Efficient decentralized monitoring of safety in distributed systems. *ICSE*, pp. 418–427. IEEE, 2004.

[27] K. Sen, A. Vardhan, G. Agha, and G. Rosu. Decentralized runtime analysis of multithreaded applications. *IPDPS*. IEEE, 2006.

[28] H. Zhu, M.B. Dwyer, and S. Goddard. Predictable runtime monitoring. In *ECRTS*, pp. 173–183. IEEE 2009.

A The language `leader`

A.1 Proof of Proposition 1

We first prove that the traffic-light monitor is correct.

Claim 1 *The opinion number of* `leader` *is at most three.*

Proof. Recall that in every execution, the output of the traffic-light opinion-maker is a multiset O of elements of $U = \{\text{red}, \text{orange}, \text{green}\}$. For O the interpretation is as follows:

$$O \in \mathbf{Y} \iff \text{red} \notin O \wedge \text{green} \in O$$

Consider an execution of the traffic-light opinion-maker on input s. Let O be the multiset of opinions decided by the processes.

– Suppose that red $\in O$. In that case, $O \in \mathbf{N}$. A process p_i decides red whenever at least two processes disagree on the identity of the leader. Even if the snapshot of p_i is a partial view of s, the fact that other processes may show up later will not change the lack of agreement. Hence, $s \notin \mathcal{L}$ and it is correct for O to be in the "no" set.

– The absence of red in the set of colors returned by the opinion-maker indicates that all participating processes agreed on a unique leader. The absence of green would however indicate that the common leader is invalid, in the sense that its identity is not the identity of a participating process. Hence, $s \notin \mathcal{L}$ and it is correct for O to be in the "no" set $\mathbf{N}$.

– The composition of the presence of green with the absence of red guarantees that all participating processes have the same leader, and that at least one process noticed that the id entity of the common leader is the identity of a participating process. Hence, it is correct for O to be in the "yes" set in the presence of green with the absence of red.

Hence, three opinions suffice to monitor the language `leader`, independently of the number of processes. □

□

We now prove that two opinions a and b are not sufficient to monitor the language `leader`, even in the case of a system with only two processes.

Claim 2 *The opinion number of* `leader` *is at least three.*

Proof. Assume, for the purpose of contradiction, that there exists a monitor (μ, M) using only two opinions, $\{a, b\}$. Notice that, in contrast to the traffic-light checker where each process writes and then snapshots once, M is an opinion-maker where processes may communicate with each other by writing and reading an arbitrary (finite) number of times. In particular, executions where a process runs *solo* will be important below. Namely, M, being wait-free, must specify how many writes and snapshots a process p should execute while not seeing any writes by any other processes, and then what opinion to produce.

Let us then consider the five following instances: $s_1 = \{(p, q)\}$, $s_2 = \{(q, q)\}$, $s_3 = \{(p, r)\}$ where $r \notin \{p, q\}$, $s_4 = \{(p, q), (q, q)\}$, and $s_5 = \{(q, q), (p, r)\}$, as depicted on Figure 2.

Instance s_2 is legal, (the leader of q is itself) while both instances s_1 and s_3 are illegal (in both cases, the identity of the leader of p is not an identity that appears in the instance). W.l.o.g., we

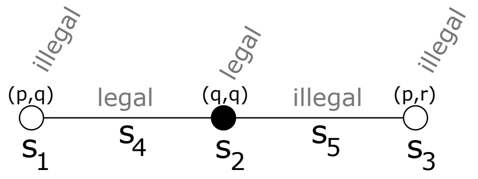

Figure 2: A partial representation of the possible inputs for monitoring `leader`.

can assume that opinion a is output by the opinion-maker when the input is s_2 (i.e., when process q runs solo with input q), while opinion b is output when the input is s_1 or s_3 (p runs solo with either input q or r). Thus, $\{a\} \in \mathbf{Y}$ and $\{b\} \in \mathbf{N}$. Let us then focus on the case in which the input is s_4. This input may be represented by an edge i.e., a 1-dimensional simplex (a set of two elements, one representing process p and its input, and the other representing process q with its input). The collection of all possible inputs actually forms a graph, e.g. a complex $\mathcal{I}$ (a collection of simplexes closed under containment — see section 6.1 for more details about topological notions.).

The fundamental theorem of wait-free read/write computability states that the final views of the processes, on all executions of a wait-free opinion-maker can be represented by a subdivision of the input complex $\mathcal{I}$ [22]. In the special case of two processes, in the case of the input s_4, $\mathcal{I}$ is simply a graph consisting of one edge, and all executions of processes p and q both with input q, the final views of the opinion-maker can be represented as a path [3] (its length depend on the number of operations of the opinion-maker). A final view of a process in this path is represented by a vertex, which can be coloured with the opinion decided by the corresponding process. Two vertices belong to the same edge, when the views of the processes correspond to the same execution of the opinion-maker. The endpoints of the path correspond to views in which one process runs solo, and decides an opinion without seeing the input of the other process. Thus, these two extremities of the subdivided edge s_4 are mapped to the two different values, a and b. Since the path is connected, at least one of its edges (representing one execution of the opinion-maker where p and q run) must also have its two vertices coloured with a and b. For more details about the structure of executions of two-process wait-free algorithms see [22] Chapter 2.

A bit more formally, the colors induce a mapping from a subdivision of the input complex $\mathcal{I}$ to the 1-dimensional sphere *complex*$(\{a,b\},2)$ [22]. Since the two extremities of the edge s_4 are mapped to the two different values a and b, every opinion-maker returns the multiset $\{a,b\}$ for at least one run of p and q on simplex s_4. As a consequence, one must have $\{a,b\} \in \mathbf{Y}$, for s_4 is legal. By the same reasoning, every opinion-maker returns the multiset $\{a,b\}$ for at least one run of p and q on input simplex s_5, which implies $\{a,b\} \in \mathbf{N}$, for s_5 is not legal. This contradiction completes the proof. □

□

A.2 Proof of Proposition 2

We show that the alternation number of `leader` is 3, in a system with at least three processes, p, q, r. Although the proof is not difficult, this serves as an illustration for the definition.

Recall that, given an instance $s = \{(\mathrm{id}_1, \ell_1), \ldots, (\mathrm{id}_k, \ell_k)\}$, we denote $\mathrm{ID}(s) = \{\mathrm{id}_1, \ldots, \mathrm{id}_k\}$ and $\mathit{val}(s) = \{\ell_1, \ldots, \ell_k\}$. Consider the following sequences of instances

$$s_1 = \{(p, q)\}, s_2 = \{(p, q), (q, q)\}, s_3 = \{(p, q), (q, q), (r, p)\},$$

where $p, q, r \in [n]$ are three distinct identities. Instance s_2 is a legal instance in which the leader of both processes p and q is q. Clearly, $s_1 \subset s_2 \subset s_3$. Moreover, the sequence s_1, s_2, s_3 alternates between legal and illegal instances. Indeed, in s_1, the leader of p is not in $\mathrm{ID}(s_1)$ (validity is violated), and, in s_3, two processes do not have the same leader (agreement is violated). Therefore, according to the definition of the alternation number (Definition 2), we have:

$$\#\mathrm{altern}(\texttt{leader}) \geq 3$$

Conversely, let $s_1 \subset \ldots \subset s_k$ be a sequence of increasing instances such that, for every $i, 1 \leq i \leq k-1$,

$$s_i \in \texttt{leader} \wedge s_{i+1} \notin \texttt{leader} \text{ or } s_i \notin \texttt{leader} \wedge s_{i+1} \in \texttt{leader}$$

Notice that $\mathit{val}(s_i) \subseteq \mathit{val}(s_{i+1})$ and $\mathrm{ID}(s_i) \subset \mathrm{ID}(s_{i+1})$, for every $i, 1 \leq i \leq k-1$. Assume that $k \geq 3$. Then consider the smallest $i, 1 \leq i \leq k-1$ for which we have $s_i \in \texttt{leader} \wedge s_{i+1} \notin \texttt{leader}$. By definition of the sequence, $i \in \{1, 2\}$. Then, either at least two distinct leaders appear in s_{i+1}, or validity is violated in s_{i+1}, i.e., $\mathit{val}(s_{i+1}) \not\subseteq \mathrm{ID}(s_{i+1})$. Instead, by construction, the instance s_i is legal: every process has the same leader whose identity is in $\mathrm{ID}(s_i)$. Therefore, since $s_i \subset s_{i+1}$, a leader ℓ that does not appear in s_i ($\ell \notin \mathit{val}(s_i)$) is elected in s_{i+1} ($\ell \in \mathit{val}(s_{i+1})$). Thus $|\mathit{val}(s_{i+1})| > |\mathit{val}(s_i)| = 1$. It thus follow that one cannot find an instance s_{i+2} that contains s_{i+1} and such that $s_{i+2} \in \texttt{leader}$. Hence, $i + 1 = k$, and, as $i \in \{1, 2\}$, the length of the sequence is at most 3. Hence,

$$\#\mathrm{altern}(\texttt{leader}) \leq 3$$

which completes the proof. □

B Proof of Corollary 1

We present a simple universal monitor (μ, M) that uses at most $n + 1$ opinions for monitoring n-processes languages. This monitor is not adaptive to the alternation number, but to the number of participating processes (the adaptiveness to the alternation number requires significantly more work). More precisely, every opinion output in any execution of the opinion-maker M in which at most k processes participate belongs to the same set of size $k + 1$. The interpretation μ is dealing with multisets of opinions taken from the set

$$U = \{(\text{green}, \ell) : 0 \leq \ell \leq \lfloor n/2 \rfloor\} \cup \{(\text{red}, \ell) : 1 \leq \ell \leq \lfloor (n+1)/2 \rfloor\}.$$

Note that $|U| = n + 1$. The partition $(\mathbf{Y}, \mathbf{N})$ of $\mathbb{N}^U$ is as follows: for any multiset $u \in \mathbb{N}^U$ we define $g = \max\{\ell : (\text{green}, \ell) \in u\}$ and $r = \max\{\ell : (\text{red}, \ell) \in u\}$. We then set: $u \in \mathbf{Y} \iff g \geq r$. Then the opinion-maker works as follows:

```
Opinion-maker $M$ at process $p$ with input $a$:
**write** $(p, a)$ in memory;
**snapshot** memory, to get $s' = \{(\mathrm{id}_1, a_1), \ldots, (\mathrm{id}_k, a_k)\}$;
**if** $s' \in \mathcal{L}$ **then** decide $(\text{green}, \lfloor \frac{|s'|}{2} \rfloor)$ **else** decide $(\text{red}, \lfloor \frac{|s'|+1}{2} \rfloor)$.
```

Let us fix an arbitrary language $\mathcal{L}$ defined over the alphabet A. Consider an execution of the opinion-maker on $s = \{(\mathrm{id}_1, a_1), \dots, (\mathrm{id}_\ell, a_\ell)\}$. That is, only processes with identities in $\mathrm{ID}(s)$ participate, and the input of process $i \in \mathrm{ID}(s)$ is the pair (i, a_i), where a_i is an element of the alphabet A. Let us assume that every participating process outputs, and let $u \in \mathbb{N}^U$ be the multiset of the opinions decided by the processes. The proof is divided in two cases according to whether $s \in \mathcal{L}$ or $s \notin \mathcal{L}$. Denote by p the last process that takes a snapshot of the memory. Notice that p observes the full input s.

Assume first that $s \in \mathcal{L}$. In this case, p decides $(\text{green}, \lfloor |s|/2 \rfloor)$, and therefore $(\text{green}, \lfloor |s|/2 \rfloor) \in u$. If the color red does not appear in u, then $u \in \mathbf{Y}$. Suppose that some process p' decides (red, ℓ'). Then, the instance s' that p' observes in the memory is such that $s' \notin \mathcal{L}$. Since the memory is read in snapshots, and since the input of the execution is s, we get that $s' \subsetneq s$. In particular, $|s'| < |s|$ and thus $\ell' = \lfloor (|s'| + 1)/2 \rfloor \leq \lfloor |s|/2 \rfloor$. Therefore, for each pair $(\text{red}, \ell) \in u$, we have $\ell \leq \lfloor |s|/2 \rfloor$. Since $(\text{green}, \lfloor |s|/2 \rfloor) \in u$, we conclude that $g \geq r$, and thus $u \in \mathbf{Y}$.

Now, let us assume that $s \notin \mathcal{L}$. As s is the instance that p observes in the memory, it decides $(\text{red}, \lfloor (|s| + 1)/2 \rfloor)$. Therefore $(\text{red}, \lfloor (|s| + 1)/2 \rfloor) \in u$. If u does not include any pair containing the color green, then $u \in \mathbf{N}$. Otherwise, suppose that some process p' decides (green, ℓ'). In that case, the instance s' that p' observes is such that $s' \in \mathcal{L}$. Again, since the memory is read in snapshots, and since the input of the execution is s with $s \notin \mathcal{L}$, we get that $s' \subsetneq s$. In particular, $|s'| < |s|$ and thus $\ell' = \lfloor |s'|/2 \rfloor \leq \lfloor (|s| - 1)/2 \rfloor < \lfloor (|s| + 1)/2 \rfloor$. Therefore, for each pair $(\text{green}, \ell) \in u$, we have $\ell < \lfloor |s|/2 \rfloor$. Since $(\text{red}, \lfloor |s|/2 \rfloor) \in u$, we conclude that $g < r$, and thus $u \in \mathbf{N}$. $\square$

C Proof of Theorem 2

We use again the equivalence between monitoring a language and solving orientation detection tasks as stated in the first bullet of the Equivalence Lemma (Lemma 1). Given a language $\mathcal{L}$ with $\#\mathrm{altern}(\mathcal{L}) = k$, we are aiming at solving wait-free an orientation-detection task with input complex $\mathcal{J}_\mathcal{L}$ using at most $k+1$ values. To describe this universal distributed algorithm, one needs to introduce the notion of *negalternation* number for the oriented complex $\mathcal{J}$, which is essentially the alternation number, restricted to counting only alternations from positive simplexes to negative simplexes. More precisely, the negalternation number of an ascending chain $s_1 \subset s_2 \subset \ldots \subset s_k$ of oriented simplexes is 1 plus the number of occurrences of an index i, $1 \leq i \leq k-1$, such that $\mathit{sign}(s_i) = +1$ and $\mathit{sign}(s_{i+1}) = -1$. The negalternation number, $\mathrm{negalt}(s)$, of an oriented simplex s is the maximum, taken over all ascending chains ending at s, of the negalternation numbers of these chains. The negalternation number of an oriented complex $\mathcal{J}$ is then defined as

$$\mathrm{negalt}(\mathcal{J}) = \max_{s \in \mathcal{J}} \mathrm{negalt}(s).$$

It follows from the definition that

$$2 \cdot \mathrm{negalt}(\mathcal{J}) - 2 \leq \#\mathrm{altern}(\mathcal{J}) \leq 2 \cdot \mathrm{negalt}(\mathcal{J}).$$

For every oriented complex $\mathcal{J}$ with $\#\mathrm{altern}(\mathcal{J}) = k$, we define an orientation-detection task T_{od} whose output complex uses at most $k+1$ values. If $\#\mathrm{altern}(\mathcal{J})$ is even, then let us set

$$U = \{(\text{green}, 1), \dots, (\text{green}, \mathrm{negalt}(\mathcal{J}) - 1), (\text{red}, 1), \dots, (\text{red}, \mathrm{negalt}(\mathcal{J}))\}\ ;$$

otherwise,

$$U = \{(\text{green}, 1), \ldots, (\text{green}, \text{negalt}(\mathcal{J})), (\text{red}, 1), \ldots, (\text{red}, \text{negalt}(\mathcal{J}))\}\ .$$

The above set U does not contain more than 2 negalt($\mathcal{J}$) values, at most two values red or green per level of negalternation, from 1 to negalt($\mathcal{J}$). By the definition, we have $2\ \text{negalt}(\mathcal{J}) \leq \#\text{altern}(\mathcal{J})+1$ except if #altern($\mathcal{J}$) is even, in which case we may have $2\ \text{negalt}(\mathcal{J}) = \#\text{altern}(\mathcal{J}) + 2$. Nevertheless, in that case, U contains $2\ \text{negalt}(\mathcal{J}) - 1$ values. It thus follows from the definition of U that $|U| \leq \#\text{altern}(\mathcal{J}) + 1$.

We are thus now focusing on the task $T_{od} = (\mathcal{J}, \mathcal{U}, \Delta_{od})$ where $\mathcal{U} = \mathit{complex}(U, n)$, and, for every $s \in \mathcal{J}$ and every $t = \{(\text{id}_1, (c_1, \ell_1)), \ldots, (\text{id}_r, (c_r, \ell_r))\} \in \mathcal{U}$ such that $\text{ID}(s) = \text{ID}(t)$, we have $t \in \Delta(s)$ if and only if $\max_{1\leq j\leq r} \ell_j \leq \text{negalt}(s)$ and

$$\begin{cases} \exists j, 1 \leq j \leq r: & (c_j, \ell_j) = (\text{green}, \text{negalt}(s)) \text{ if } \mathit{sign}(s) = +1; \\ \exists j, 1 \leq j \leq r: & (c_j, \ell_j) = (\text{red}, \text{negalt}(s)) \text{ and} \\ & \forall i, 1 \leq i \leq r,\ (c_i, \ell_i) \neq (\text{green}, \text{negalt}(s)) \text{ if } \mathit{sign}(s) = -1. \end{cases}$$

Let us verify that T_{od} is indeed an orientation-detection task. To that end, let $s, s' \in \mathcal{J}$, $t, t' \in \mathcal{U}$ be simplexes such that $t \in \Delta(s)$, $t' \in \Delta(s')$, $\text{ID}(s) = \text{ID}(t)$, $\text{ID}(s') = \text{ID}(t')$ and $\mathit{sign}(s) \neq \mathit{sign}(s')$. If $\text{negalt}(s) \neq \text{negalt}(s')$, then $\mathit{val}(t) \neq \mathit{val}(t')$ since, for some color $c \in \{\text{green}, \text{red}\}$, the pair

$$(c, \max\{\text{negalt}(s), \text{negalt}(s')\})$$

is included in one of the multisets but not in the other. Otherwise, $\text{negalt}(s) = \text{negalt}(s') = \ell$, and, since $\mathit{sign}(s) = -\mathit{sign}(s')$, it follows from the definition of Δ_{od} that only one of the two multisets $\mathit{val}(t)$ and $\mathit{val}(t')$ contains (green, ℓ).

A wait-free distributed algorithm M for T_{od} then performs as follows:

```
Orientation-detector $M$:
write $(i, s_i)$ in memory;
snapshot memory, to get $s' \in \mathcal{J}$;
if $\mathit{sign}(s') = +1$ then decide $(\text{green}, \text{negalt}(s'))$
else decide $(\text{red}, \text{negalt}(s'))$.
```

We prove that distributed algorithm M is correct. Let $s \in \mathcal{J}$ be an oriented simplex. Assume that the processes participating in the distributed algorithm are all processes with identities in ID(s) and that they all decide. For every process p_i, $i \in \text{ID}(s)$, denote by (c_i, ℓ_i) the value decided by p_i and denote by t the simplex representing the decisions. At least one process i_0 reads s in memory. Every other process reads some simplex $s' \subseteq s$, and thus decides some pair (c, ℓ) where $c \in \{\text{green,red}\}$ and $\ell = \text{negalt}(s') \leq \text{negalt}(s)$. If $\mathit{sign}(s) = +1$, then process i_0 decides $(\text{green}, \text{negalt}(s))$. Hence, $(\text{green}, \text{negalt}(s), i_0) \in t$ and thus $t \in \Delta(s)$. Instead, if $\mathit{sign}(s) = -1$, then process i_0 decides $(\text{red}, \text{negalt}(s))$. Now, every $s'_i \subseteq s$ with $\text{negalt}(s'_i) = \text{negalt}(s)$ have $\mathit{sign}(s'_i) = -1$. Hence, every process p_i that decides (c_i, ℓ_i) with $\ell_i = \text{negalt}(s)$ satisfies $c_i = \text{red}$. Hence $(\text{green}, \text{negalt}(s)) \notin t$, and therefore, $t \in \Delta(s)$.

Let $\mathcal{L}$ be a language. We showed above that a wait-free solvable orientation-detection task $T_{od} = (\mathcal{J}_{\mathcal{L}}, \mathcal{U}, \Delta_{od})$, where $\mathcal{J}_{\mathcal{L}}$ is the complex induced by $\mathcal{L}$, and $\mathcal{U} = \mathit{complex}(U, n)$ for a set U of size at most $\#\text{altern}(\mathcal{J}_{\mathcal{L}}) + 1$, can be defined. Therefore, since $\#\text{altern}(\mathcal{J}_{\mathcal{L}}) = \#\text{altern}(\mathcal{L})$ by

Lemma 2, we get that the first bullet of the Equivalence Lemma (Lemma 1) implies that $\mathcal{L}$ can be monitored with at most #altern$(T) + 1$ opinions.

The opinion-maker for monitoring $\mathcal{L}$ is the distributed algorithm M described above. A generic interpretation is described in the proof of the first bullet of Lemma 1. A more specialized interpretation μ is as follows. Given

$$S = \{(c_j, \ell_j) : c_j \in \{\text{green,red}\}, \ell_j \in \mathbb{N}, j = 1, \ldots, r\},$$

let

$$\text{maxlevel} = \max_{j=1,\ldots,r} \ell_j.$$

Then,

$$S \in \mathbf{Y} \iff (\text{green,maxlevel}) \in S$$

To see that the pair (μ, M) is indeed a monitor for $\mathcal{L}$, consider an instance s. By definition of $\mathcal{J}_T$, s is a simplex of $\mathcal{J}_{\mathcal{L}}$ oriented $+1$ if and only if s is a legal instance. Assuming that every process with IDs in $\text{ID}(s)$ decides, let t be the simplex representing the decisions in some execution of M with input s, and let S be the multiset $S = \mathit{val}(t)$. By definition of Δ_{od}, for every $(c, \ell) \in S$, $\ell \leq \text{negalt}(\sigma)$ and $(c, \text{negalt}(s)) \in S$. Moreover, $(\text{green}, \text{negalt}(s)) \in S$ if and only if $\mathit{sign}(s) = +1$. Therefore $(\text{green}, \max_{(c,\ell)\in S} \ell) \in S$, i.e., $S \in \mathbf{Y}$, if and only if $\mathit{sign}(s) = +1$ or, equivalently, $s \in \mathcal{L}$. $\square$

D Proof of the equivalence lemma (Lemma 1)

To establish the first statement, let $\mathcal{L}$ be a language defined over some alphabet A, and assume that there exists a wait-free distributed algorithm M solving the orientation-detection task $T = (\mathcal{J}_{\mathcal{L}}, \mathcal{U}, \Delta)$ using values in some set U of size k. We claim that (μ, M) is a monitor for $\mathcal{L}$ where the interpretation μ is defined as follows:

$$\mathbf{Y} = \{\mathit{val}(u) \mid u \in \Delta(r),\ r \in \mathcal{J}_{\mathcal{L}} \text{ and } \mathit{sign}(r) = +1\}\ .$$

and

$$\mathbf{N} = \{\mathit{val}(u) \mid u \in \Delta(r),\ r \in \mathcal{J}_{\mathcal{L}} \text{ and } \mathit{sign}(r) = -1\}$$

If $\mathcal{L}$ is on n processes, then $\mathbf{N}$ and $\mathbf{Y}$ are composed of multisets with size at most n. For every instance s, we have

$$s \in \mathcal{L} \iff \mathit{sign}(s) = +1 \iff \forall u \in \Delta(s), \mathit{val}(u) \in \mathbf{Y}\ .$$

Similarly:

$$s \notin \mathcal{L} \iff \forall u \in \Delta(s), \mathit{val}(u) \in \mathbf{N}\ .$$

Therefore, if $s \in \mathcal{L}$, then $\mathit{val}(M(s)) \in \mathbf{Y}$. Similarly, if $s \notin \mathcal{L}$, then $\mathit{val}(M(s)) \in \mathbf{N}$. As a consequence, (μ, M) is indeed a monitor for $\mathcal{L}$.

To establish the second statement of the lemma, let $\mathcal{J}$ be a well-formed oriented complex, let $k \geq 1$, and assume that no orientation-detection task for $\mathcal{J}$ is wait-free solvable with k values. Then, let us consider the language $\mathcal{L}_{\mathcal{J}}$, and assume, for the purpose of contradiction, that there exists a monitor (μ, M) for $\mathcal{L}_{\mathcal{J}}$ using opinions in U, with $|U| = k$. Let $A = \{-1, +1\} \times V(\mathcal{J})$

denote the alphabet over which $\mathcal{L}_{\mathcal{J}}$ is defined. Recall that the interpretation μ specifies a partition $(\mathbf{Y}, \mathbf{N})$.

In an execution of the opinion-maker M, processes receive as input element of A and decides values in U. The multiset of the decided values is in $\mathbf{Y}$ if the elements of A assigned to the processes form a legal instance of $\mathcal{L}_{\mathcal{J}}$ and in $\mathbf{N}$ otherwise. M thus solves the task $T = (\mathcal{I}, \mathcal{O}, \Delta)$ where

$$\mathcal{I} = \mathit{complex}(A, n),\ \mathcal{O} = \mathit{complex}(U, n)$$

and Δ is defined as follows. For $J \subseteq [n]$, let us set:

$$Y[J] = \{u \in \mathit{complex}(U, n) : \mathrm{ID}(u) = J\ \wedge\ \mathit{val}(u) \in \mathbf{Y}\}$$

and

$$N[J] = \{u \in \mathit{complex}(U, n) : \mathrm{ID}(u) = J\ \wedge\ \mathit{val}(u) \in \mathbf{N}\}\ .$$

For every $s \in \mathit{complex}(A, n)$ with $\mathrm{ID}(s) = J$, we then have:

$$\Delta(s) = \begin{cases} Y[J] & \text{if } s \in \mathcal{L}_{\mathcal{J}} \\ N[J] & \text{otherwise.} \end{cases}$$

By definition of $\mathit{complex}(A, n)$, for each simplex $s \in \mathcal{J}$, there are two simplexes s^+ and s^- in $\mathit{complex}(A, n)$ except that every vertex of s^+ is additionally labeled $+1$ and every vertex of s^- is additionally labeled -1. Let $\mathcal{J}^+$ denote the complex identical to $\mathcal{J}$ except that vertex of $\mathcal{J}^+$ is additionally labeled $+1$. Note that $\mathcal{J}^+$ is a subcomplex of $\mathit{complex}(A, n)$.

Let $r, r' \in \mathcal{J}^+$ with $\mathit{sign}(r) \neq \mathit{sign}(r')$, that is the simplexes corresponding to r and r' have opposite orientation in $\mathcal{J}$, say $\mathit{sign}(r) = +1 = -\mathit{sign}(r')$. Hence, $r \in \mathcal{L}_{\mathcal{J}}$ while $r' \notin \mathcal{L}_{\mathcal{J}}$. Therefore, the output of the opinion-maker is in $\mathbf{Y}$ for r and $\mathbf{N}$ for r'. Thus, the opinion-maker M maps r and r' to two simplexes u and u' of $\mathit{complex}(U, n)$ satisfying $\mathit{val}(u) \neq \mathit{val}(u')$. Therefore, M also solves an orientation-detection task with input complex $\mathcal{J}$ and output complex $\mathit{complex}(U, n)$. The fact that M is a wait-free distributed algorithm solving T is thus a contradiction with the fact that no orientation-detection tasks for $\mathcal{J}$ are wait-free solvable with k values. Hence, the language $\mathcal{L}_{\mathcal{J}}$ requires at least $k + 1$ opinions to be monitored. □

E Proof of Lemma 2

Let $\mathcal{L}$ be a distributed language defined over the alphabet A. The fact that the oriented complex $\mathcal{J}_{\mathcal{L}}$ associated with it has the same alternation number as $\mathcal{L}$ directly follows from the definition of $\mathcal{J}_{\mathcal{L}}$. To see why, recall that $\mathcal{J}_{\mathcal{L}} = \mathit{complex}(A, n)$ and, for every simplex $s \in \mathcal{J}_{\mathcal{L}}$, we have $\mathit{sign}(s) = +1$ if and only if $s \in \mathcal{L}$.

Let now $\mathcal{J}$ be a well-formed oriented complex. We show that the language $\mathcal{L}_{\mathcal{J}}$ associated with it has alternation number at most $\#\mathrm{altern}(\mathcal{J}) + 1$. Recall that language $\mathcal{L}_{\mathcal{J}}$ is defined over the alphabet $A = \{+1, -1\} \times V(\mathcal{J})$, and an instance $s = \{(i_1, (\sigma_1, v_1)), \dots, (i_k, (\sigma_k, v_k))\}$ is legal if and only if for some $\sigma \in \{-1, +1\}$, $\sigma_i = \sigma$ for every $i, 1 \leq i \leq k$ and $\{(i_1, v_1), \dots, (i_k, v_k)\}$ is a simplex of $\mathcal{J}$ with orientation σ. Given such an instance s, we will denote by $\widetilde{s}$ the simplex $\widetilde{s} = \{(i_1, v_1)), \dots, (i_k, v_k)\}$. Let $S = s_1 \subset \dots \subset s_{\ell+1}$ be a sequence of instances with alternating legality. Let k be the index of the largest legal instance in the sequence. Note that $k \in \{\ell, \ell + 1\}$.

As $s_k \in \mathcal{L}_{\mathcal{J}}$, for any pair of elements $(i,(\sigma,v))$, $(i',(\sigma',v'))$ in s_k, $\sigma=\sigma'=\mathit{sign}(\widetilde{s}_k)$. Therefore, for any $j, 1 \le j \le k$ and any element $(i'',(\sigma'',v''))$ in s_j, $\sigma''=\mathit{sign}(\widetilde{s}_k)$ as $s_j \subseteq s_k$. Similarly, $\widetilde{s}_j \subseteq \widetilde{s}_k$ and it thus follows that $\widetilde{s}_j \in \mathcal{J}$. Consequently, if instance s_j is illegal, this is because the common sign (i.e., $\mathit{sign}(\widetilde{s}_k)$) associated with each element of s_j is different from the orientation of the simplex $\widetilde{s}_j$ in $\mathcal{J}$. This implies that the sequence $\widetilde{S}=\widetilde{s}_1 \subset \ldots \subset \widetilde{s}_k$ of increasing simplexes of $\mathcal{J}$ alternates between simplexes of opposite orientations. The length of $\widetilde{S}$ is upper-bounded by the alternation number of $\mathcal{J}$, i.e., $k \le \#\mathrm{altern}(\mathcal{J})$. Since $k \in \{\ell, \ell+1\}$, we conclude that $\#\mathrm{altern}(\mathcal{L}_{\mathcal{J}}) \le \#\mathrm{altern}(\mathcal{J})+1$. □

F Proof of Theorem 1

Let $0 \le k < n$. We exhibit a language for n processes with alternation number $k+1$ that cannot be monitored with $k+1$ or less opinions. We do not construct such a language directly but rather rely on the correspondence between monitoring languages and solving orientation-detection tasks, as stated in the second bullet of the Equivalence Lemma (cf. Lemma 1). We define and orient a complex $\mathcal{J}$ of dimension $n-1$ with alternation number k. We next show that any orientation-detection task with input complex $\mathcal{J}$ cannot be solved wait-free with less than k value. Then, by the Equivalence Lemma, the language $\mathcal{L}_{\mathcal{J}}$ induced by $\mathcal{J}$ requires at least $k+1$ opinions to be monitored. To complete the proof, we show that $\#\mathrm{altern}(\mathcal{L}_{\mathcal{J}})=k+1$. That is, language $\mathcal{L}_{\mathcal{J}}$ has alternation number $k+1$ and requires at least $k+1$ opinions to be monitored.

We start with the definition of an oriented complex $\mathcal{J}_k$ for k processes with $\#\mathrm{altern}(\mathcal{J}_k)=k$. We establish that solving wait-free *any* orientation-detection task with input complex $\mathcal{J}_k$ requires at least $k+1$ values. We then extend the complex $\mathcal{J}_k$ to form an oriented complex $\mathcal{J}$ for n processes without changing the alternation number, i.e., $\#\mathrm{altern}(\mathcal{J})=\#\mathrm{altern}(\mathcal{J}_k)$. Since $\mathcal{J}_k$ is a sub-complex of $\mathcal{J}$, solving an orientation-detection task with input $\mathcal{J}$ also requires at least $k+1$ values.

Let s_{k-1} and s'_{k-1} be two simplexes of dimension $k-1$ properly colored with identities $\{1,\ldots,k\}$. Moreover, s_{k-1} and s'_{k-1} share a face of dimension $(k-2)$ that we denote s_{k-2}. The input complex $\mathcal{J}_k$ is the complex induced by $s_{k-1} \cup s'_{k-1}$. That is, $\mathcal{J}_k$ consists of the simplexes s_{k-1}, s'_{k-1} and each of their faces.

Fix a sequence $(s_i)_{0 \le i \le k-2}$ of simplexes of increasing dimension included in the common face s_{k-2}:

$$s_0 \subset s_1 \subset \ldots \subset s_{k-2}$$

Note that each simplex s_i has dimension i. We assign an orientation to each simplex s of $\mathcal{J}_k$ inductively:

$$\forall s \in \mathcal{J}_k, \mathit{sign}(s) = \begin{cases} +1 & \text{if } s = s_0; \\ -\mathit{sign}(s_i) & \text{if } s \subseteq s_{i+1} \text{ and } s \not\subseteq s_i \text{ for } 0 \le i < n-1; \\ -\mathit{sign}(s_{k-2}) & \text{if } s \subsetneq s'_{k-1} \text{ and } s \not\subseteq s_{k-2}; \\ \mathit{sign}(s_{k-2}) & \text{if } s = s'_{k-1}. \end{cases}$$

The construction is illustrated in Figure 3 for the case $k=3$. Each simplex is labeled with its orientation. To avoid cluttering the figure, identities are represented by the color of each vertex.

Note that the two simplexes of dimension $(k-1)$, s_{k-1} and s'_{k-1}, have opposite orientations. Also, notice that the sequence of simplexes $s_0,\ldots,s_{k-1}$ is such that $s_i \subset s_{i+1}$ and $\mathit{sign}(s_i) =$

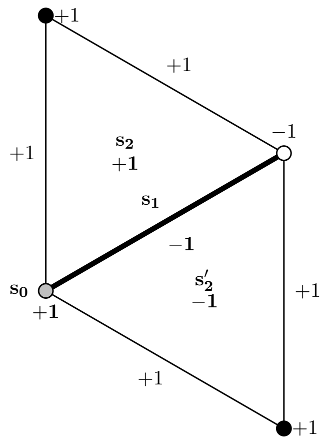

Figure 3: Oriented input complex for the orientation-detection task T_3.

$-sign(s_{i+1})$ for each $i, 0 \leq i < k-1$. Hence, the alternation number of $\mathcal{J}_k$ is $\#\text{altern}(\mathcal{J}_k) = k$.

There are orientation-detection tasks with input $\mathcal{J}_k$ that are wait-free solvable, e.g., the task $(\mathcal{J}_k, \mathcal{J}_k, Id)$ where Id is the identity map. Next, we consider a set U and a wait-free solvable orientation-detection task $T_k = (\mathcal{J}_k, \mathcal{U}, \Delta)$ with input complex $\mathcal{J}_k$, where $\mathcal{U} = complex(U, n)$. We establish that $|U| \geq k+1$.

Claim 3 *If the orientation-detection task $T_k = (\mathcal{J}_k, \mathcal{U}, \Delta)$, where $\mathcal{U} = complex(U, n)$, is wait-free solvable, then $|U| \geq k+1$.*

To establish the claim, let A be a wait-free distributed algorithm for T_k. Without loss of generality, we assume that A is a *full-information* distributed algorithm in *normal form* [3, 22]. Full-information means that every process writes all it knows each time it performs a write operation. A distributed algorithm is in normal form if each process p_i proceeds in a sequence of rounds. In each round, p_i writes to the register $M[i]$ and reads the whole memory M. We restrict our attention to a subset of the possible executions of A, namely, *immediate snapshot* (IS) executions. An IS execution proceeds in rounds. Each round is specified by a non-empty set of processes, which are *active* in the round. A round consists of concurrent writes performed by each active process followed by concurrent snapshots of the shared memory by each active processes. IS executions capture the computing power of the wait-free read-write shared memory model [23].

We recall some facts on representing distributed computations by combinatorial objects. Our exposition follows the framework developed in previous works, e.g., [3].The *distributed algorithm complex* $\mathcal{P}$ of A gives a representation of the set of all IS executions of A. Each vertex $v \in V(\mathcal{P})$ represents the final state of some process at the end of some IS execution of A with input some simplex $s \in \mathcal{J}_k$. Each vertex v is thus colored with an identity $\text{ID}(v) \in [k]$, a state $state(v)$, and a value $\mu(v) \in U$ such that is $\mu(v)$ the value decided by the process with identity $\text{ID}(v)$ in some IS execution in which its final state is $state(v)$. A set $\sigma = \{v_1, \ldots, v_\ell\} \subseteq V(\mathcal{P})$ is a simplex of $\mathcal{P}$ if it is properly colored with ID and there is an IS execution of A in which the final state of the process with identity $\mathrm{ID}(v_i)$ is $state(v_i)$, for each $i, 1 \leq i \leq \ell$. Given a simplex $s \in \mathcal{J}_k$, the *IS-complex for* s, denoted $\mathcal{P}(s)$ is the sub-complex of $\mathcal{P}$ containing each simplex that corresponds to an IS execution with input s. For every $s \in \mathcal{J}_k$, $\mathcal{P}(s)$ is a subdivision of s, and in particular $\mathcal{P}(s)$ is a $dim(s)$*-pseudomanifold* [3, 22]. A ℓ-dimensional (sub-)complex $\mathcal{C}$ is a ℓ-pseudomanifold if it is pure for dimension ℓ and every $(\ell-1)$-dimensional simplex of $\mathcal{C}$ is contained in one or two ℓ-dimensional simplexes of $\mathcal{C}$. For every simplex $\sigma \in \mathcal{P}$, the *carrier* of σ $carrier(\sigma)$ is the simplex $s \in \mathcal{J}_k$ of smallest dimension such that $\sigma \in \mathcal{P}(s)$.

For each simplex $\sigma = \{v_1, \ldots, v_\ell\} \in \mathcal{P}$, denote by $val(\sigma) = \{\mu(v_1), \ldots, \mu(v_\ell)\}$ the multiset of values output in the IS execution of A corresponding to σ. As A solves an orientation-detection task with input complex $\mathcal{J}_k$, we have:

$$\{\mu(\sigma) \mid \sigma \in \mathcal{P} \text{ and } sign(carrier(\sigma)) = +1\} \cap \{\mu(\sigma) \mid \sigma \in \mathcal{P} \text{ and } sign(carrier(\sigma)) = -1\} = \emptyset$$

The proof relies on the following property which is satisfied by the subdivision $\mathcal{P}$ and the coloring μ for every $i, 0 \leq i \leq k-1$:

$\mathbf{P}_i$: There exists a set of $i+1$ distinct colors $\{c_0, \ldots, c_i\}$ such that

$$|\{\sigma \mid carrier(\sigma) = s_i \text{ and } \mu(\sigma) = \{c_0, \ldots, c_i\}\}|$$

is odd.

Property $\mathbf{P}_i$ means that there exists an odd number of simplexes of dimension i in the subdivided complex $\mathcal{P}(s_i)$ colored with the same $i+1$ distinct colors by μ. The property also holds if we replace the last element of the sequence $s_0 \subset \ldots \subset s_{k-1}$ by s'_{k-1}, i.e.,

$\mathbf{P}'_{k-1}$: There exists a set of k distinct colors $\{c'_0, \ldots, c'_{k-1}\}$ such that

$$|\{\sigma \mid carrier(\sigma) = s'_{n-1} \text{ and } \mu(\sigma) = \{c'_0, \ldots, c'_{k-1}\}\}|$$

is odd.

Assume that both properties $\mathbf{P}_{k-1}$ and $\mathbf{P'}_{k-1}$ are true. By construction of the orientation of $\mathcal{J}_k$, $sign(s_{k-1}) = -sign(s'_{k-1})$. Hence, for every $\sigma, \sigma' \in \mathcal{P}$ such that $carrier(\sigma) = s_{k-1}$ and $carrier(\sigma') = s'_{k-1}$, $\mu(\sigma) \neq \mu(\sigma')$. Therefore, $\{c'_0, \ldots, c'_{k-1}\} \neq \{c_0, \ldots, c_{k-1}\}$, from which we conclude that $|U| \geq k+1$, as desired.

We next show by induction that $\mathbf{P}_i$ is true for every $i, 0 \leq i \leq k-1$ and that $\mathbf{P}'_{k-1}$ is also satisfied. For $i = 0$, let c_0 be the color assigned to a vertex with carrier s_0 by μ. Since there is a unique vertex with carrier s_0 in $\mathcal{P}$, $\mathbf{P}_0$ is true.

Fix $i, 0 \leq i \leq k-2$. Assume that $\mathbf{P}_i$ is satisfied for some set of $i+1$ distinct colors $\{c_0, \ldots, c_i\}$. We consider the sub-complex $\mathcal{P}(s_{i+1})$ and its coloring induced by μ. The proof is similar to the proof of Sperner's Lemma. However, since the coloring of $\mathcal{P}(s_{i+1})$ may not be a Sperner coloring as the vertexes on the boundaries of $\mathcal{P}(s_{i+1})$ may be colored arbitrarily, we cannot directly apply this Lemma.

Following the proof of Sperner's Lemma [21], we construct a graph $G = (V, E)$. We associate a vertex denoted $v(s)$ to each simplex $s \in \mathcal{P}(s_{i+1})$. Then,

$$V = \{v(s) \mid s \in \mathcal{P}(s_{i+1}) \text{ and } dim(s) = i+1\} \cup \{u\}$$

where u is an external vertex, which is not associated with any simplex of $\mathcal{P}(s_{i+1})$.

For every pair s, s' of $(i+1)$-dimensional simplexes in $\mathcal{P}(s_{i+1})$, there is an edge between $v(s)$ and $v(s')$ if and only if s and s' share an i-dimensional face colored with $\{c_0, \ldots, c_i\} : \forall v(s) \neq v(s') \in V$,

$$\{v(s), v(s')\} \in E \iff \mathit{val}(s \cap s') = \{c_0, \ldots, c_i\}$$

Similarly, for every $(i+1)$-dimensional simplex s in $\mathcal{P}(s_{i+1})$, there is an edge between u and $v(s)$ if and only if an i-dimensional face s' of s is colored with $\{c_0, \ldots, c_i\}$ and the dimension of $\mathit{carrier}(s')$ is i. In other words, s has an i-dimensional face s' in the boundary of $\mathcal{P}(s_{i+1})$ and $\mu(s') = \{c_0, \ldots, c_i\} : \forall v(s) \in V$,

$$\begin{aligned} \{v(s), u\} \in E \quad \iff \quad & \exists s' \subset s \mid \mathit{dim}(\mathit{carrier}(s')) = i \\ & \text{and } \mathit{val}(s') = \{c_0, \ldots, c_i\} \end{aligned}$$

Let $s \in \mathcal{P}(s_{i+1})$ be a simplex of dimension $(i+1)$. It follows from the definition of G and the fact that $\mathcal{P}(s_{i+1})$ is an $(i+1)$-pseudomanifold that:

$$\mathit{deg}(v(s)) = \begin{cases} 1 & \text{if } \{c_0, \ldots, c_i\} \subsetneq \mathit{set}(\mathit{val}(s)); \\ 2 & \text{if } \{c_0, \ldots, c_i\} = \mathit{set}(\mathit{val}(s)); \\ 0 & \text{otherwise.} \end{cases}$$

where $\mathit{set}(m)$ denotes the set formed by the elements in the multiset m.

Suppose now that s has an i-dimensional face s' in the boundary of $\mathcal{P}(s_{i+1})$, i.e., $\mathit{dim}(\mathit{carrier}(s')) = i$. Assume in addition that $\mathit{val}(s) = \{c_0, \ldots, c_i\}$. By construction of the orientation of $\mathcal{J}_k$, for each i-dimensional face $t \neq s_i$ of s_{i+1}, $\mathit{sign}(t) = -\mathit{sign}(s_i)$. Hence, two simplexes with carrier s_i and t, respectively, cannot be colored the same way by μ. Moreover, by the induction hypothesis, there exists a simplex of dimension i in $\mathcal{P}(s_i)$ colored $\{c_0, \ldots, c_i\}$ by μ. Therefore, $\mathit{carrier}(s) = s_i$, and since the number of i-dimensional simplexes that belong to $\mathcal{P}(s_i)$ and that are colored $\{c_0, \ldots, c_i\}$ by μ is odd (Property $\mathbf{P}_i$), it follows that:

$$\mathit{deg}(u) \text{ is odd.}$$

Note that this also holds in the case $s_{i+1} = s'_{k-1}$, since by construction, for every $(k-2)$-dimensional face $t \neq s_{k-2}$ of s'_{k-1}, $\mathit{sign}(t) = -\mathit{sign}(s_{k-2})$.

Since $\sum_{v \in V} \mathit{deg}(v)$ is even, there must exist an odd number of $(i+1)$-dimensional simplexes s in $\mathcal{P}(s_{i+1})$ such that $\{c_0, \ldots, c_i\} \subsetneq \mathit{set}(\mathit{val}(s))$. Therefore, for some color $c_{i+1} \in U \setminus \{c_0, \ldots, c_i\}$, the number of simplexes $s \in \mathcal{P}(s_{i+1})$ such that $\mathit{val}(s) = \{c_0, \ldots, c_{i+1}\}$ is odd, which completes the proof of $\mathbf{P}_{i+1}$ and $\mathbf{P}'_{n-1}$ (if $i = n-2$).

This completes the proof of Claim 3.

Essentially, to complete the proof of Theorem 1, we extend task T_k to obtain an orientation-detection task T for n processes whose input complex $\mathcal{J}$ contains the input complex $\mathcal{J}_k$ of T_k. Our construction does not change the alternation number and preserves the orientation of $\mathcal{J}_k$. So $\#\text{altern}(\mathcal{J}) = \#\text{altern}(\mathcal{J}_k)$, and any wait-free distributed algorithm A for T can also be used by k processes for T_k, and therefore requires at least $k+1$ values by Claim 3.

Claim 4 *There exists an $(n-1)$-dimensional oriented complex $\mathcal{J}$ with alternation number $\#\text{altern}(\mathcal{J}) = k$ such that for any orientation-detection task $T = (\mathcal{J}, \mathcal{U}, \Delta)$, where $\mathcal{U} = \mathit{complex}(U, n)$, if T is wait-free solvable, then $|U| \geq k+1$.*

We define the $(n-1)$-dimensional oriented input complex $\mathcal{J}$ as follows. Let $\{v_{k+1},\ldots,v_n\}$ and $\{v'_{k+1},\ldots,v'_n\}$ be two sets of vertexes properly colored with identities $\{k+1,\ldots,n\}$, i.e., $\forall i,j,k+1\le i,j\le n$, $v_i \neq v'_j$ and $\mathrm{ID}(v_i)=\mathrm{ID}(v'_i)=i$. Denote by s_{n-1} and s'_{n-1}, respectively, the simplexes $s_{n-1}=s_{k-1}\cup\{v_{k+1},\ldots,v_n\}$ and $s'_{n-1}=s'_{k-1}\cup\{v'_{k+1},\ldots,v'_n\}$.

$\mathcal{J}$ is the complex induced by $s_{n-1}\cup s'_{n-1}$. That is, simplex s is included in $\mathcal{J}$ if and only if $s\subseteq s_{n-1}\vee s\subseteq s'_{n-1}$. $\mathcal{J}$ is thus a complex of dimension $n-1$; every simplex in $\mathcal{J}$ is properly colored with identities $\subseteq\{1,\ldots,n\}$. Moreover, $\mathcal{J}_k$ is a sub-complex of $\mathcal{J}$. For each simplex $s\in\mathcal{J}_k$, denote by $sign_{\mathcal{J}_k}(s)$ the orientation assigned to s in the definition of $\mathcal{J}_k$.

By construction, for every $v_i, k+1\le i\le n$ and $v'_j, k+1\le j\le n$, no simplex s of $\mathcal{J}$ includes both v_i and v'_j. Notice also that each simplex $s\in\mathcal{J}$ that does not contain any vertex v_i or v'_i for any $i, k+1\le i\le n$ is a simplex included in the sub-complex $\mathcal{J}_k$. The complex $\mathcal{J}$ is oriented as follows:

$$\forall s\in\mathcal{J}, sign(s)=\begin{cases} sign_{\mathcal{J}_k}(s) & \text{if } s\in\mathcal{J}_k;\\ sign(s_{k-1}) & \text{if } \exists v_i, k+1\le i\le n \mid v_i\in s;\\ sign(s'_{k-1}) & \text{if } \exists v'_i, k+1\le i\le n \mid v'_i\in s.\end{cases}$$

The input complex for 4 processes obtained by extending of the input complex of T_3 is depicted in Figure 4.

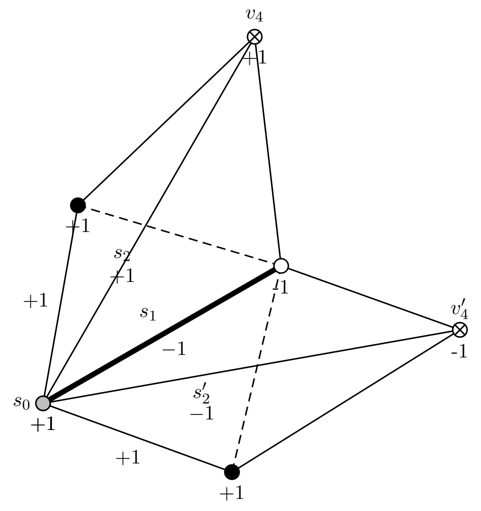

Figure 4: Extending T_3 for 4 processes. Every simplex that includes vertex v_4 is oriented $+1$, the same orientation as s_2. Similarly, every simplex that includes vertex v'_4 is oriented -1 as s'_2. Other simplexes keep their original orientation, as described in Figure 3.

Let U be an arbitrary set and let $\mathcal{U}$ denote the complex $complex(U,n)$. Consider an orientation-detection task $T=(\mathcal{J},\mathcal{U},\Delta)$ with input complex $\mathcal{J}$. Since $\mathcal{J}_k$ is a sub-complex of $\mathcal{J}$, and each simplex $s\in\mathcal{J}_k$ has the same orientation in $\mathcal{J}_k$ and $\mathcal{J}$, every wait-free distributed algorithm A for T is also a wait-free solution for the orientation-detection task $T_k=(\mathcal{J}_k,\mathcal{U},\Delta)$. It thus follows from Claim 3 that A requires at least $k+1$ colors, i.e., $|U|\ge k+1$.

We next show that the alternation number of $\mathcal{J}$ is #altern$(\mathcal{J}) = k$. To that end, consider an arbitrary sequence $t = t_0 \subset t_1 \subset \ldots \subset t_m$ of simplexes in $\mathcal{J}$. For any sequence $\tau = \tau_0 \ldots \tau_j$ of simplexes, denote by $\beta(\tau)$ the number of changes of orientation in the sequence, i.e.,

$$\beta(\tau) = |\{i \mid 0 \leq i < j \text{ and } sign(\tau_i) = -sign(\tau_{i+1})\}|$$

Let t_ℓ denote the simplex in t of largest dimension that is included in $\mathcal{J}_k$. By construction of $\mathcal{J}$, $t_{\ell+1}$ must contain either a vertex v_i or a vertex v'_i, for some $i, k+1 \leq i \leq n$. Assume without loss of generality that $v_i \in t_{\ell+1}$. Since $t_{\ell+1} \subset \ldots \subset t_m$, $v_i \in t_j$ for every $j, \ell+1 \leq j \leq m$ from which it follows that $sign(t_{\ell+1}) = \ldots = sign(t_m)$. Hence, $\beta(t) = \beta(t_0, \ldots, t_{\ell+1}) \leq \ell + 1$.

Since the dimension of $\mathcal{J}_k$ is $k-1$, $\ell \leq k-1$. If $\ell < k-1$, $\beta(t) = \beta(t_0, \ldots, t_{\ell+1}) \leq k-1$. Otherwise, $\ell = k-1$ and thus the dimension of t_ℓ is $k-1$. The complex $\mathcal{J}_k$ contains two simplexes of dimension $k-1$: s_{k-1} and s'_{k-1}. As $t_\ell \subset t_{\ell+1}$ and a vertex v_i, for some $i, k+1 \leq i \leq n$, is included in $t_{\ell+1}$, it follows from the construction of $\mathcal{J}$ that $t_\ell = s_{k-1}$. Thus, $sign(s_{k-1}) = sign(t_\ell) = sign(t_{\ell+1})$, from which we conclude that $\beta(t) = \beta(t_0, \ldots, t_{\ell+1}) = \beta(t_0, \ldots, t_\ell) \leq \ell = k-1$. This complete the proof that for every sequence $t_0 \subset \ldots \subset t_m$ of simplexes in $\mathcal{J}$, the number of times two consecutive simplexes have opposite orientation is at most $k-1$. Therefore, #altern$(\mathcal{J}) \leq k$. As $\mathcal{J}_k$ is a subcomplex of $\mathcal{J}$, and each simplex $s \in \mathcal{J}_k$ has the same orientation in $\mathcal{J}_k$ and $\mathcal{J}$, #altern$(\mathcal{J}) \geq k$. Hence #altern$(\mathcal{J}) = k$.

This completes the proof of Claim 4.

We use the Equivalence Lemma (Lemma 1) to complete the proof of the theorem. By Claim 4, there exists an $(n-1)$-dimensional oriented complex $\mathcal{J}$ with alternation number k such that any orientation-detection task with input $\mathcal{J}$ requires at least $k+1$ values to be wait-free solvable. Therefore, it follows from the second bullet of the Equivalence Lemma (Lemma 1) that language $\mathcal{L}_{\mathcal{J}}$ requires $k+1$ opinions to be monitored.

We next show that #altern$(\mathcal{J}_{\mathcal{L}}) = k+1$. Recall that $\mathcal{L}_{\mathcal{J}}$ is defined over the alphabet $A = \{-1, +1\} \times V(\mathcal{J})$. For an instance

$$t = \{(i_1, (\sigma_1, v_1)), \ldots, (i_\ell, (\sigma_\ell, v_\ell))\},$$

let $\widetilde{t}$ denote the simplex $\widetilde{t} = \{(i_1, v_1), \ldots, (i_\ell, v_\ell)\}$. Let $T = t_0 \subset \ldots \subset t_k$ a sequence of instances defined as follows. For each $i, 0 \leq i \leq k-1$, $\widetilde{t_i} = s_i \in \mathcal{J}$ and for each element $(id, (\sigma, v))$ of t_i, $\sigma = sign(s_i)$ if i and $k-1$ have the same parity and $\sigma = -sign(s_i)$ otherwise. t_k is such that $\widetilde{t_k}$ is a k-dimensional simplex of $\mathcal{J}$ containing s_{k-1} and there are two elements $(id, (\sigma, v))$, $(id', (\sigma', v'))$ in t_k, $\sigma \neq \sigma'$. Note that instance t_k is illegal and, for $i : 0 \leq i \leq k-1$, instance t_i is legal if and only if i and $k-1$ have the same parity. Hence, the sequence T alternates between legal and illegal instances from which we have #altern$(\mathcal{L}_J) \geq k+1$. As #altern$(\mathcal{J}) = k$, it follows from Lemma 2 that #altern$(\mathcal{L}_{\mathcal{J}}) = k+1$, which completes the proof of the theorem for $k < n$.

For $k = n$, Claim 4 immediately follows from Claim 3 and the language $\mathcal{L}_{\mathcal{J}}$ has alternation number $n = k$. It requires at least $n+1$ opinions to be monitored by the Equivalence Lemma. □