Perfect failure detection with very few bits.

Pierre Fraigniaud, Sergio Rajsbaum, Corentin Travers, Petr Kuznetsov, Thibault Rieutord

Perfect Failure Detection with Very Few Bits

Pierre Fraigniaud, Sergio Rajsbaum, Corentin Travers, Petr Kuznetsov, Thibault Rieutord

Perfect Failure Detection with Very Few Bits*

Pierre Fraigniaud[1], Sergio Rajsbaum[2], Corentin Travers[3], Petr Kuznetsov[4], and Fabien Rieutord[4]

[1]IRIF and U. Paris Diderot, France
[2]Instituto de Matemáticas, UNAM, Mexico
[3]LaBRI, U. Bordeaux, France
[4]Telecom Paristech, France

September 13, 2016

Abstract

A *failure detector* is a distributed oracle that provides each process with a module that continuously outputs an estimate of which processes in the system have failed. The *perfect* failure detector provides accurate and eventually complete information about process failures. We show that, in asynchronous failure-prone message-passing systems, perfect failure detection can be achieved by an oracle that outputs at most $\lceil \log \alpha(n) \rceil + 1$ bits per process in n-process systems, where α denotes the inverse-Ackermann function. This result is essentially optimal, as we also show that, in the same environment, no failure detector outputting a constant number of bits per process can achieve perfect failure detection.

1 Introduction

Failure detectors have influenced research and development of fault-tolerant distributed systems for over 20 years, since their introduction in two seminal papers [2, 3]. A *failure detector* is an abstraction layer that provides each process with information about which other processes have crashed. The concept of failure detector provides a modular approach of distributed computing and an elegant framework which yields two orthogonal but interacting working projects: developing portable algorithms on top of failure detectors, and developing efficient failure detector implementations in various message passing and shared memory settings. This concept has been very successful in a wide variety of settings, including network communication protocols, group membership protocols, and algorithms for solving consensus, atomic commit, broadcast, mutual exclusion, leader election, as well as several other services (see Section 1.2). More generally, the failure detector abstraction has fostered the theoretical understanding of failures, and of their effect in distributed computing. Indeed, failure detectors abstract away details of the system (e.g., the message delivery times on each link, the process speeds, etc.), by focussing only on extracting process failure information. Given a failure detector, one can then investigate what are the distributed computing tasks that are solvable with the information provided by this failure detector layer (irrespective of the underlying network on top of which the failure detector is implemented).

In a nutshell, failure detectors provide a formal framework to tackle questions such as: How much, and what kind of information about failures is necessary to solve a given distributed computing task?

At the one end of the spectrum, the concern is the *minimum information about failures* needed to solve a given task, e.g., notably, the *consensus* problem. Various *weakest* failure detector classes for consensus have been identified, that show that the question of *how much* information about failures is needed to solve

*This research has been carried out within the framework of ECOS Nord (Project M12M01).

a problem is subtle. The weakest failure detectors enabling to solve consensus are all equivalent, that is, given any of these failure detectors, one can build any other such failure detector. Yet, they seem to provide very different kind of information, presented in very different forms. For instance, the failure detector Ω outputs the identity of a single process at each process [2]. This identity is such that, eventually, all the correct processes are provided with the same identity, which is the identity of a correct process. In contrast, the failure detector $\Diamond S$ outputs a set of process identities at each process [3]. These identities are the ones of suspected processes, and are such that, eventually, they include all the processes that have crashed, and there is a correct process whose identity is included in none of the sets. These two failure detectors are equivalent (they both are weakest failure detectors enabling to solve consensus in an asynchronous system where a majority of processes are correct).

At the other end of the spectrum, the concern is failure detectors that provide *perfectly accurate information about failures*. Remarkably, also at this end of the spectrum, there are failure detectors that provide "the same information" about failures, but they do so in a very different way. This is raising the question of what is the *amount* of information provided by failure detectors. Consider for example the failure detector classes $\mathcal{P}$ and ψ^t. A failure detector $\mathcal{P}$ provides each process with a set of identities of processes that are suspected to have failed [3]. These sets are such that non-faulty processes are never suspected, and all faulty processes are eventually suspected by each process. In a system where at most t processes can crash, a failure detector ψ^t outputs an integer at each process [17]. These integers are such that they are at most the number of processes that crash, and, eventually, they are all equal to the number of processes that have actually crashed. While $\mathcal{P}$ and ψ^t are quite similar qualitatively as they both provide perfectly accurate information about failures, they are quantitatively quite different: one provides sets of identities, while the other provides integers in a bounded range of values.

In this paper, we initiate the study of how many bits should be provided to each process by a failure detector to ensure specific knowledge about the failure pattern, or to solve a given task. We start our investigation by tackling this question at the latter end of the spectrum, namely, the case of a failure detector that guarantees perfectly accurate information about failures. Specifically, in this paper, we tackle the following question: *how many bits should be provided to each process by a failure detector that guarantees perfectly accurate information about failures?*

1.1 Contributions

We describe a new failure detector, called *micro-perfect*, denoted μP, which outputs at most $\lceil \log \alpha(n) \rceil + 1$ bits at each process, in asynchronous failure-prone message-passing systems with n processes, where α denotes the inverse-Ackermann function. We show that μP is equivalent to the perfect failure detector. This result is essentially optimal, as we also show that, in asynchronous failure-prone message-passing systems, no failure detector outputting a constant number of bits at each process can achieve perfect failure detection.

For establishing both our lower and upper bounds, we use techniques from well-quasi-ordering theory [13]. This important tool in logic and computability has a wide variety of applications [15]. Here we proceed to explore the depth of the connection of well-quasi-orderings with distributed computing, stemming from an essential difficulty when dealing with processes that may crash. In fault-tolerant computing, when a process considers a list L of local states of other processes, it may well be the case that its view is incomplete, e.g., the actual global state is L' with $L \subset L'$, because it is possible that processes in $L' \setminus L$ are delayed. This bares resemblance to well-quasi-ordering theory, which studies words over alphabets, and the sub-words that can be obtained by deleting some symbols of each word. We show that fault-tolerant computing does not only bare resemblance to well-quasi-ordering theory, but that well-quasi-ordering theory is inherently present in some aspects of fault-tolerant computing.

More specifically, for the lower bound, a key ingredient is Higman's lemma [11], which essentially says that if $w^{(1)}, w^{(2)}, \ldots$ is an infinite sequence of words over some *finite* alphabet Σ, then there exist indices $i < j$ such that $w^{(i)}$ can be obtained from $w^{(j)}$ by deleting some of its letters. We show how to use Higman's lemma to prove that no failure detector outputting a constant number of bits at each process can achieve perfect failure detection.

For the upper bound, i.e., for the design of the micro-perfect failure detector μP, we use a combination of failure detector techniques, with the notion of *distributed encoding* of the integers recently introduced in [6]. A distributed encoding of the integers is a distributed structure that encodes each positive integer n by a word $w^{(n)} = w_1^{(n)}, \ldots, w_n^{(n)}$ over some (non-necessarily finite) alphabet Σ, such that no proper sub-words of $w^{(n)}$ can be interpreted as the distributed encoding of n' with $n' < n$. In [6], using well-quasi-ordering theory, it is proved that the first n integers can be distributedly encoded using words on an alphabet with letters on $\lceil \log \alpha(n) \rceil + 1$ bits, where α is a function growing at least as slowly as the inverse-Ackerman function. We explain how to use this encoding to prove that there exists a failure detector μP outputting $\lceil \log \alpha(n) \rceil + 1$ bits at each process, which achieves perfect failure detection. A companion technical report [7] contains the proofs and some additional material.

1.2 Related Work

We refer to [8] for a recent survey on the failure detector abstraction. In this section, we just survey work closely related to our paper.

In [17], two failure detectors are introduced, which output an integer that approximates the number of crashed processes. More precisely, a query to a failure detector of the class ψ^y returns an integer that is always between $t - y$ and the number of processes that crash during the execution (where t is the maximum of processes that can crash, and $0 \leq y \leq t$). More precisely, for any time τ, the output returned by a query issued at time τ is at most $\max(t - y, f^\tau)$ where f^τ is the number of processes that have crashed at time τ. Furthermore, there is a time τ' from which the output returned by any query issued at any time τ'' after time τ' is equal to $\max(t - y, f^{\tau''})$. The class $\Diamond\psi^y$ relaxes ψ^y by allowing the properties defining ψ^y to be satisfied only eventually. It is proved that the classes ψ^y and $\Diamond\psi^y$ are respectively equivalent[1] to the classes ϕ^y and $\Diamond\phi^y$ of [16]. A failure detector of the class ϕ^y provides the processes with a query primitive which has a set X of processes as parameter, and which returns a boolean answer. When $|X|$ is too small (or too big), the invocation of the query for X by a process returns systematically *true* (resp., $false$). Otherwise, namely, when $t - y < |X| \leq t$, $0 \leq y \leq t$, the query returns *true* only if all the processes in X have crashed. Moreover, if all the processes of X have crashed, and a process repeatedly issues the query, then it eventually obtains the answer *true*. Notice that ϕ^0 provides no information about failures, while ϕ^t is equivalent to a perfect failure detector. In the follow-up paper [17], the relation of the failure detector Ω_z to set agreement is studied, including relations with respect to the failure detector $\Diamond S_x$.

A failure detector class whose output is binary has been introduced in [9] to solve non-blocking atomic commit. This class, called *anonymously perfect failure detectors*, and denoted by $?\mathcal{P}$, is defined as follows. Each process has a flag (initially equal to $false$) that is eventually set to *true* if and only if a process has crashed (the identity of the crashed process is not necessarily known, hence the name "anonymous"). The definition of $?\mathcal{P}$ has been extended in [17] to take into account the fact that k processes have crashed (instead of just one). This class, denoted $?\mathcal{P}k$, provides each process with a flag that is eventually set to *true* if and only if at least k processes have crashed (observe that $?\mathcal{P}$ is $?\mathcal{P}1$). An interesting question raised in [17] is the issue of additivity of failure detectors. It is known that combining two failure detectors may enable solving consensus, while none of them is individually strong enough to enable solving consensus. In this paper, we aim at quantifying such phenomenon, by considering the number of bits provided to each process by the failure detector.

The notion of *well-quasi-ordering* (wqo) is a "frequently discovered concept", as already pointed out by Kruskal [13] in 1972. One important application of wqo is providing *termination arguments* in decidability results [1]. Indeed, thirteen years after publishing his undecidability result, Turing [21] proposed the now classic method of proving program termination using so-called "bad sequences", with respect to a wqo. In the setting of wqo, the problem of *bounding* the length of bad sequences is of utmost interest as it yields upper bounds on terminating program executions. Hence, the interest in *algorithmic aspects* of wqos has

[1] We stress that, in the literature, by "equivalent" it is meant that, given any failure detector of one class, it is possible to build a failure detector of the other class, and it is understood that "both provide the same information on failures" (see, e.g., [17]).

grown recently as witnessed by the amount of work collected in [19]. For more applications and related work on wqos, including rewriting systems, tree embeddings, lossy channel systems, and graph minors, see recent work [10, 19]. The notion of distributed encoding of the integers was proposed in [6] to show that every one-shot system specification can be wait-free runtime monitored non-deterministically using only three opinions.

2 The model

Our results are stated in the classical model used for investigating failure detectors. Specifically, we consider an asynchronous crash-prone message-passing system consisting of n processes denoted by $p_1, \ldots, p_n$. Each process p_i has a unique identity $i = \mathsf{id}(p_i)$, and the total number n of processes is known to each of the processes. Each pair of processes $\{p_i, p_j\}$ is connected by a reliable, yet asynchronous channel. That is, any message sent by p_i to p_j is eventually received by p_j but there are no upper bounds on the time to transfer that message. Channels are reliable in the sense that they do not alter, duplicate or create messages. An arbitrary large number of processes can fail, by crashing, as long as at least one process remains correct. When a process crashes, it permanently stops functioning, that is, it does not execute any more steps of computation (including sending and receiving messages).

2.1 Failure detectors

For modeling failure detectors, we assume the existence of a global clock, with non-negative integer values. This clock is however not accessible to the processes. Let $\Pi = \{p_1, \ldots, p_n\}$. Each process in Π may fail by crashing. A process that crashes halts taking steps, and never recovers. A *failure pattern* is a function $\mathcal{F} : \mathbb{N} \to 2^{\Pi}$ that specifies which are the processes that have crashed by time $\tau \in \mathbb{N}$. Let $\mathsf{faulty}(\mathcal{F}) = \bigcup_{\tau \in \mathbb{N}} \mathcal{F}(\tau)$ be the set of processes that fail in the failure pattern $\mathcal{F}$. The set of processes that do not fail is $\mathsf{correct}(\mathcal{F}) = \Pi \setminus \mathsf{faulty}(\mathcal{F})$. When there is no ambiguity on the underlying failure pattern $\mathcal{F}$, we say that a process p_i is *correct* if $p_i \in \mathsf{correct}(\mathcal{F})$ and *faulty* if $p_i \in \mathsf{faulty}(\mathcal{F})$. An *environment* is a set of failure patterns. In this paper, as specified before, all failure patterns in which at least one process is correct can occur.

A failure detector [3] is a distributed device that provides each process with some information on the failure pattern. Each process can *query* the failure detector, and each query returns a value in some (potentially infinite) range $\mathcal{R}$ that depends on the failure detector. The outputs of a failure detector during an execution is described by a failure detector *history*, which is a function $H : \Pi \times \mathbb{N} \to \mathcal{R}$ that maps each pair process-time to a value in $\mathcal{R}$. The value returned by the failure detector to process p_i at time τ is $H(p_i, \tau)$. A failure detector D with range $\mathcal{R}$ associates a non-empty set of histories with range $\mathcal{R}$ to every failure pattern. The set of histories corresponding to a failure pattern $\mathcal{F}$ is denoted by $D(\mathcal{F})$. That is, $D(\mathcal{F})$ is a collection of functions of the form $H : \Pi \times \mathbb{N} \to \mathcal{R}$, and when the failure pattern is $\mathcal{F}$, the behavior of the failure detector coincides with some history $H \in D(\mathcal{F})$. For instance, the failure detector Ω [2] has range $\{0, 1\}$, and guarantees that it eventually outputs 1 at a single correct process, and 0 at every other processes. That is, for every failure pattern $\mathcal{F}$, the history $H : \Pi \times \mathbb{N} \to \{0, 1\} \in \Omega(\mathcal{F})$ if and only if there exists $p_i \in \mathsf{correct}(\mathcal{F})$, and $\tau \in \mathbb{N}$ such that, for every $\tau' \geq \tau$, $H(p_i, \tau') = 1$ and $H(p_j, \tau') = 0$ for every $j \neq i$.

The so-called *perfect* failure detector P [3] provides a list of processes that have crashed to each process. The failure detector P does not make any mistake, in the sense that no process is declared crashed before it has failed. Moreover, it is eventually complete, in the sense that its output at every process eventually matches the set of faulty processes. More formally, the range of P is 2^{Π}, and, for every failure pattern $\mathcal{F}$, the history $H : \Pi \times \mathbb{N} \to 2^{\Pi}$ belongs to $P(\mathcal{F})$ if and only if the following two properties are satisfied: (Accuracy) for every time τ and process p_i, $H(p_i, \tau) \subseteq \mathcal{F}(\tau)$; and (Completeness) there exists a time τ such that, for every $\tau' \geq \tau$ and process p_i, $H(p_i, \tau') = \mathcal{F}(\tau')$.

Similarly, the *eventual perfect* failure detector $\Diamond P$ [3] is identical to the failure detector P except that the accuracy property only holds eventually. Formally, for every failure pattern $\mathcal{F}$, $H : \Pi \times \mathbb{N} \to 2^{\Pi}$ belongs to $\Diamond P(\mathcal{F})$ if and only if there exists $\tau \in \mathbb{N}$ such that, for every $\tau' \geq \tau$ and every process p_i, $H(p_i, \tau') = \mathcal{F}(\tau')$.

2.2 Protocols and executions

A *distributed protocol* $\mathcal{A}$ consists of n local algorithms $\mathcal{A}(p_1), \ldots, \mathcal{A}(p_n)$, one per process. An *execution* is a sequence of *steps*. During a step, every process p_i acts according to its local algorithm. First, it performs some local computation, and then it performs one of the following five actions: (1) sending a message to some process, (2) receiving a (possibly empty) set of messages, (3) querying the failure detector, (4) receiving an external input or, (5) sending an external output. In a reception step performed by process p_i, since the communication channels are asynchronous, the set of messages might be empty even if a message has been previously send to p_i and not yet received by it. Receiving external input (resp., sending external outputs) are actions enabling to specify protocols that implement, or emulate failure detectors. External inputs correspond to queries to the emulated failure detector. As a result of such a query, an external output is eventually sends, which corresponds to the result of the query.

An *execution* of a protocol $\mathcal{A}$ using failure detector D in environment $\mathcal{E}$ is a tuple $exec = (\mathcal{F}, H, S, T)$ where $\mathcal{F}$ is a failure pattern in $\mathcal{E}$, H is a failure detector history in $D(\mathcal{F})$, S is a sequence of steps of $\mathcal{A}$, and T is a strictly increasing sequence of clock ticks in $\mathbb{N}$. S is called a *schedule*, and the ith step $S[i]$ in S is taken at time $T[i]$. A tuple $exec = (\mathcal{F}, H, S, T)$ defines an execution of $\mathcal{A}$ if and only if the following conditions are satisfied: (1) every correct process takes infinitely many steps in S, (2) no processes take a step after they have crashed, (3) the sequence T and S are either both finite with the same length, or are both infinite, (4) no messages are loss, i.e., if a process performs infinitely many receive step, it eventually receives all messages that were sent to it, (5) if the ith step $S[i]$ is a failure detector query by process p_j that returns d, then $d = H(p_j, T[i])$, i.e., the failure detector queries return values that are consistent with the history H, and (6) the steps taken in S are consistent with the protocol $\mathcal{A}$. Formalizing the above conditions is straightforward but requires care and heavy notation. We refer to [4, 12] for such a formalization.

2.3 Comparing failure detectors

A protocol $\mathcal{A}$ that implements a failure detector D receives queries as external inputs, and produces responses in the range of D. Since computing a response may entail sending/receiving messages as well as local computations, there might be some delay between the time τ at which $\mathcal{A}$ receives a query, and the time τ' at which $\mathcal{A}$ produces a response d to that query. The correctness condition taken from [12] requires that d must be a legal output for D at some point in time between τ and τ'. Hence the implementation $\mathcal{A}$ of D behaves as an atomic failure detector, for which responses to queries are given instantaneously. More precisely, a protocol $\mathcal{A}$ implements a failure detector D using a failure detector D', or, for short, *emulates* D using D', in environment $\mathcal{E}$ if, for every failure pattern $\mathcal{F} \in \mathcal{E}$, and for every execution $exec = (\mathcal{F}, H', S, T)$ of $\mathcal{A}$ where $H' \in D'(\mathcal{F})$ the following hold. Let H_Q and H_R be the histories of external inputs (queries) and outputs (responses to queries) in Execution *exec*. A query occurs at process p_i at time τ if $H_Q(p_i, \tau) = query$. Similarly, a response occurs at time τ at process p_i if $H_R(p_i, \tau) = d$, where d is a value in the range of D. The following three properties must be fulfilled: (1) for every correct process p_i, and every integer $i \geq 1$, if the ith query at process p_i occurs at time τ, then a response occurs at p_i at some time $\tau' > \tau$ (the ith query and the ith response occurring at the same process p_i are said to be *matching*); (2) for every process p_i, and every integer $i \geq 1$, if the ith response occurs at time τ, then the ith query occurs at p_i at some time $\tau' < \tau$; (3) there exists a failure detector history $H \in D(\mathcal{F})$ such that, for every process p_i, and every times τ_1, τ_2, if $H_Q(p_i, \tau_1) = query$, $H_R(p_i, \tau_2) = d$, and this query/response pair is matching, then $d = H(p_i, \tau)$ for some time $\tau \in [\tau_1, \tau_2]$.

We are now ready to describe how to compare failure detectors. Let D and D' be two failure detectors. We say that D *is at least as weak as* D' in environment $\mathcal{E}$, denoted by $D \leq_{\mathcal{E}} D'$, if there is a protocol that implements D using D' in environment $\mathcal{E}$. Then D and D' are said *equivalent* in environment $\mathcal{E}$ if $D \leq_{\mathcal{E}} D'$ and $D' \leq_{\mathcal{E}} D$. These notions are motivated by the fact that if a failure detector D can be used to solve some task T in some environment $\mathcal{E}$ then every failure detector D' such that $D \leq_{\mathcal{E}} D'$ can be used as well to solve the task T. For example, consensus can be solved using Ω [18]. If $\Omega \leq_{\mathcal{E}} D$, then, in $\mathcal{E}$, one can compose a protocol $\mathcal{B}$ that implements Ω using D with a protocol $\mathcal{A}$ solving consensus using Ω. Finally, a failure detector D is said to be a *weakest* failure detector for a task T in environment $\mathcal{E}$ if and only if (1)

there is a protocol that solves T using D in environment $\mathcal{E}$ and, (2) for every failure detector D' that can be used to solve T in $\mathcal{E}$, we have $D \leq_{\mathcal{E}} D'$. For example, it has been shown [2] that Ω is a weakest failure detector for consensus in the *majority* environment, i.e., the environment in which every failure pattern $\mathcal{F}$ satisfies $|\mathsf{faulty}(\mathcal{F})| < \frac{n}{2}$. Also, Ω is the weakest failure detector to implement eventual consistency [5].

3 Perfect failure detection requires $\omega(1)$ bits per process

In this section, we show that any failure detector emulating the perfect failure detector P must output values whose range depends on the size n of the system.

Theorem 3.1. *A failure detector that outputs a constant number of bits at each process cannot emulate the perfect failure detector P.*

3.1 Preliminaries

Two key ingredients in the proof of Theorem 3.1 are *Ramsey's Theorem* and some elements of *well-quasi-order* theory.

Ramsey's Theorem might be seen as a generalization of the pigeonhole principle. The statement of its finite version, which we are going to use in the proof is recalled below. A n-subset is a subset of size n and coloring α is a function that maps each n-subset to some element of a set of size c.

Theorem 3.2 (Ramsey's Theorem)**.** *For all natural integers n, m, and c, there exists an integer $g(n, m, c)$ with the following property. For every set S of size at least $g(n, m, c)$, and any coloring of the n-subsets of S with at most c colors, there is some subset C of S of size m that has all of its n-subsets colored the same color.*

We recall next some basic notions of well-quasi order theory. Let A a (finite or infinite) set, and let $\preceq$ be a binary relation over A. A (finite or infinite) sequence $a_1, a_2, \ldots, a_\ell$ of elements of A is *good* if there exists two indices $i < j$ such that $a_i \preceq a_j$. Otherwise, for every $i < j$, $a_i \not\preceq a_j$, the sequence is said to be *bad*. The pair $(A, \preceq)$ is a *well-quasi-order* (*wqo* for short), if (1) $\preceq$ is transitive and reflexive, and (2) every infinite sequence of elements of A is good.

A finite sequence $a_1, a_2, \ldots, a_k$ of elements of A is called a *word*. Let A^* denote the set of words, and let $\preceq_*$ be the sub-word relation over A^* induced by the relation $\preceq$. That is, $a = a_1, \ldots, a_k \preceq_* b = b_1, \ldots, b_\ell$ if and only if $k \leq \ell$ and there exists a strictly increasing mapping $m : [1, k] \to [1, \ell]$ such that $a_i \preceq b_{m(i)}$, for every i, $1 \leq i \leq k$. Higman's lemma essentially states that every bad sequence of words in $(A^*, \preceq_*)$ is finite whenever $(A, \preceq)$ is a wqo. More specifically:

Lemma 3.3 (Higman's lemma [11])**.** *If $(A, \preceq)$ is a well-quasi-order, then so is $(A^*, \preceq_*)$.*

For the purpose of establishing Theorem 3.1, we are interested in $(\Sigma^*, =^*)$ where Σ is a *finite* set, and $=^*$ denotes the sub-word relation based on the equality relation. That is, for any two words $a = a_1, \ldots, a_k, b = b_1, \ldots, b_\ell$, $a =^* b \in \Sigma^*$ if and only if there exists a strictly increasing map $m : [1, k] \to [1, \ell]$ such that $a_i = b_{m(i)}$, for every $i, 1 \leq i \leq k$.

Since Σ is finite, $(\Sigma, =)$ is a wqo. It thus follows from Higman's lemma that $(\Sigma^*, =^*)$ is also a wqo. Hence, every bad sequence over $(\Sigma^*, =^*)$ is finite. We are interested in the maximal length of such bad sequences. Of course, if no further assumption is made, bad sequences of arbitrary lengths can be constructed. However, in the case of *controlled* bad sequences, (coarse) upper bounds on the length of bad sequence have been established:

Theorem 3.4 (Length function Theorem [20])**.** *Let $L_{\Sigma^*}(d)$ be the maximal length of bad sequences $x_0, x_1, x_2, \ldots$ over $(\Sigma^*, =^*)$ such that $|x_i| \leq f^i(d) = f(f(...f(d)))$ for $i = 0, 1, 2, \ldots$ If the* control function *f is primitive-recursive, then the* length function *$L_{\Sigma^*}(d)$ is bounded by a function in $\mathcal{F}_{\omega^{|\Sigma|-1}}$.*[2]

[2]The function classes $\mathcal{F}_\alpha$ are the elementary-recursive closure of the functions F_α, which are the ordinal-indexed levels of the Fast-Growing Hierarchy [14]. Multiply-recursive complexity starts at level $\alpha = \omega$, i.e., Ackermannian complexity, and stopping just before level $\alpha = \omega^\omega$, i.e., Hyper-Ackermannian complexity.

In the proof below, we will construct a sequences $x = x_1, x_2, \ldots$ over $(\Sigma^*, =^*)$ where $|x_i| = i$, for every $i = 1, 2, \ldots$, i.e., we will restrict our attention to sequences controlled by the successor function $f : n \to n+1$, whose initial element has length 1. By the Length function Theorem, there is a bound depending solely on the cardinality of Σ on the length of every such sequence that is bad:

Corollary 3.5. *Let Σ be a finite set. There exists an integer L_{Σ^*} with the following property: Every bad sequence $x = x_1, x_2, \ldots$ over $(\Sigma^*, =^*)$ such that $|x_i| = i$ for every $i = 1, 2, \ldots$ has length at most L_{Σ^*}.*

3.2 Overview of the proof of Theorem 3.1

Let Σ be a finite set. We are going to show that there exists an integer N such that no failure detector with range Σ can emulate the perfect failure detector P in a N-process system.

For the sake of contradiction, let us assume that there is a failure detector X whose range is Σ and an algorithm $\mathcal{T}_{X \to P}$ that emulates P in a system $\Pi = \{p_1, \ldots, p_N\}$ consisting in N processes. We aim at constructing two executions *exec* and *exec*$'$ indistinguishable up to a certain point to a subset of the processes. However, the executions have different failure patterns and, by leveraging the indistinguishably of *exec* and *exec*$'$ from the perspective of some processes, we show that in one execution, a correct process is erroneously suspected by the emulated perfect failure detector.

A process p not to be able to distinguish between *exec* and *exec*$'$, it must in particular receive the same sequence of outputs from the underlying failure detector X in both executions. Let $\mathcal{F}$ be a failure pattern in which p is correct. Since the range of X is finite, in any valid history $H \in X(\mathcal{F})$, there exists a symbol in the range Σ of X that is output infinitely often at process p. Hence, by appropriately scheduling the queries to failure detector X, we can concentrate on executions in which the failure detector output is constant at each process. Furthermore, we consider only failure patterns in which each faulty process fails initially, e.g., before taking any step in the emulation algorithm.

Each such execution *exec* can be associated with a word $x_{exec} \in \Sigma^*$, namely the word formed by the failure detector constant outputs at each correct process. More precisely, the rth symbol of x_{exec} is the (constant) output of the failure detector X at the rth correct process, where processes are ordered by increasing ids. By considering executions with increasing sets of correct processes, we obtain a sequence $x = x_1, x_2, \ldots$ of words of Σ^*. If the system is sufficiently large and, thus, the induced sequence x of words in Σ^* is sufficiently long, x is good by The Length function Theorem. Hence, we are able to exhibit two words $x_{i_1}, x_{i_2}, i_1 < i_2$ where x_{i_1} is a sub-word of x_{i_2}. By construction, these words represent in fact the outputs of X at the correct processes in two executions $exec_1$ and $exec_2$ respectively with two different sets of correct processes.

Hence, at some processes, the output of X is the same in both execution, although the failure patterns differ. This is however not sufficient to conclude that $exec_1$ and $exec_2$ are indistinguishable from the perspective of some processes. Indeed, a common symbol in x_1 and x_2 may be output in $exec_1$ and $exec_2$ by processes with distinct ids. We resolve this issue by leveraging Ramsey's Theorem. We show that in a sufficiently large system, there is a subset S of processes of size strictly larger than L_{Σ^*} for which the outputs of X are essentially id-oblivious: sets of correct processes in S of the same size are provided with the same failure-detector outputs.

In more detail, for any set of processes C, let $\mathcal{F}_C$ denote the failure pattern in which the set of correct processes is C and every faulty process crashes at time 0. Let $L = L_{\Sigma^*} + 1$, where L_{Σ^*} is the bound in Corollary 3.5. Provided that the total number of processes is large enough, we show that there exists a sequence $x = x_1, \ldots, x_L$ of words of Σ^* such that $|x_i| = i$ for every $i, 1 \leq i \leq L$ with the following property. For every i-subset $C \subseteq S$, $1 \leq i \leq L$, there is a failure detector history in $X(\mathcal{F}_C)$ in which for each r, $1 \leq r \leq i$ the failure detector outputs infinitely often the rth symbol of x_i to the rth process (where the processes are ordered by increasing ids).

3.3 Proof of Theorem 3.1

In the following, we will denote by k the cardinality of Σ and L will denote the upper bound in Corollary 3.5 on the length of bad sequences over Σ^* controlled by the successor function and whose initial element has

length 1.

Let $m_0, \ldots, m_L$ be integers satisfying

$$\begin{aligned} m_0 &= g(1, m_1, k) \\ m_1 &= g(2, m_2, k^2) \\ \vdots &= \quad \vdots \\ m_i &= g(i+1, m_{i+1}, k^{i+1}) \\ \vdots &= \quad \vdots \\ m_L &= g(L+1, m_{L+1}, k^{L+1}) \\ m_{L+1} &= L+1 \end{aligned}$$

where g is the function appearing in Theorem 3.2. Let $i, 0 \leq i \leq L$. The equations satisfied by the integers $m_0, \ldots, m_L$ imply the following property according to Ramsey's Theorem, for every $i, 1 \leq i \leq N$:

> For any set S_i of size m_i, independently of the way its $(i+1)$-subsets are colored with k^{i+1} colors, one can find a subset $S_{i+1} \subseteq S_i$ of size m_{i+1} such that each of its $(i+1)$-subset has the same color.

We set the number of processes $N = m_0$, i.e., $\Pi = \{p_1, \ldots, p_N\}$. We define a coloring function α that associates each subset of processes with a color. Let $C = \{q_1, \ldots, q_\ell\} \subseteq \Pi$ a subset of the processes, where $\mathsf{id}(q_i) < \mathsf{id}(q_{i+1})$, for each $i, 1 \leq i < \ell$. The color $\alpha(C)$ of C is a word in Σ^ℓ. Let $\mathcal{F}_C$ denote the failure pattern in which the set of correct processes is C and every faulty process crashes at time 0, and let $H \in X(\mathcal{F}_C)$ some valid history for that failure pattern. Since the range Σ of X is finite, for each process q_i, there is a symbol $s_i \in \Sigma$ that is output infinitely often at process q_i in history H. We set $\alpha(C) = s_1, \ldots, s_\ell$.

We next construct inductively a sequence of subsets of processes $\Pi \supseteq S_1 \supseteq S_2 \supseteq \ldots \supseteq S_{L+1}$ and a sequence $x = x_1, \ldots, x_{L+1}$ of words of Σ^* as follows:

- Base case $i = 1$. The function α defines a k-coloring of the singletons of Π. Since $N = m_0 = g(1, m_1, k)$ is large enough, there exists according to Ramsey's theorem a subset S of Π of size m_1 such that every singleton of S has the same color. We set $S_1 = S$ and x_1 that color.
- Induction step $1 < i \leq L+1$. Let $i, 1 < i \leq L+1$. Note that by construction $|S_{i-1}| = m_{i-1}$, and we have $m_{i-1} = g(i, m_i, k^i)$. Observe also that the coloring α maps each i-subset of S_{i-1} to a word in Σ^i, e.g., the i-subsets are colored using at most k^i colors. Therefore, it follows from Ramsey's Theorem that there is a set $S \subseteq S_{i-1}$ of size m_i in which every i-subset has the same color. We set $S_i = S$ and let x_i be this common color.

$x = x_1, \ldots, x_{L+1}$ is a sequence of words of Σ^* with $|x_i| = i$, for every $i, 1 \leq i \leq L+1$. Since $|x| > L$, x must be a good sequence according to Corollary 3.5. Hence, there exists i_1, i_2, $1 \leq i_1 < i_2 \leq L+1$ such that x_{i_1} is a sub-word of x_{i_2}. More precisely, let $x_{i_1} = s_1^{(i_1)}, \ldots, s_{i_1}^{(i_1)}$ and $x_{i_2} = s_1^{(i_2)}, \ldots, s_{i_2}^{(i_2)}$. There exists a strictly increasing map $m : [1, i_1] \to [1, i_2]$ with the following property: $s_j^{(i_1)} = s_{m(j)}^{(i_2)}$ for every $j, 1 \leq j \leq i_1$.

Let $C_2 = \{q_1, \ldots, q_{i_2}\}$ be a i_2-subset of S_{L+1}, where $\mathsf{id}(q_j) < \mathsf{id}(q_{j+1})$ for every $j, 1 \leq j < i_2$. Let $C_1 = \{q'_1, \ldots, q'_{i_1}\}$ where $q'_j = q_{m(j)}$, for every $j, 1 \leq j \leq i_1$. As m is strictly increasing, $\mathsf{id}(q'_j) < \mathsf{id}(q'_{j+1})$ for every $j, 1 \leq j < i_1$. C_1 and C_2 are respectively i_1-subset and i_2-subset of S_{L+1}. As $S_{i_1} \supseteq S_{i_2} \supseteq S_{L+1}$, $\alpha(C_1) = x_{i_1}$ and $\alpha(C_2) = x_{i_2}$.

It follows from the definition of α that for every $j \in \{1, 2\}$, there exists an history $H_j \in X(\mathcal{F}_{C_j})$ in which the output of X at the rth process (order by increasing ids) of C_j is infinitely often the rth symbol in x_{i_j} (namely, $s_r^{(i_j)}$), for every r, $1 \leq r \leq i_j$.

For each $j \in \{1, 2\}$, we define an execution $exec_j$ of the emulation protocol with failure pattern $\mathcal{F}_{C_j}$, and failure detector history H_j as follows: In $exec_j$, every process $p \in C_j$ keeps performing queries to the

(emulated) failure detector P. The processes take steps one after the other, in a round robin fashion. When process p is scheduled, it takes one step. If this step is a receive step, then it receives every message with destination p that has not yet been delivered. Moreover, we schedule every query to the failure detector X made by p in such a way that the returned value is $s_r^{(i_j)}$, where r is the rank of $\mathsf{id}(p)$ among the ids of the processes in C_j. Note that this is always possible because $s_r^{(i_j)}$ is infinitely often the value output by X at process p. Specifically, whenever p is activated in the round robin schedule, and its next step is a query to X, we schedule p such that it makes its step at the next time τ at which $H(p,\tau) = s_r^{(i_j)}$.

We now consider another execution of the emulation protocol, denoted $exec_2'$. Recall that $C_1 = \{q_1', \ldots, q_{i_1}'\}$ is a subset of C_2. Execution $exec_2'$ is similar to $exec_2$ except that, before some time τ^* defined later, only the processes $q_1', \ldots, q_{i_1}'$ take step. More precisely, the failure pattern in $exec_2'$ is $\mathcal{F}_{C_2}$, and the failure detector history of X is H_2. Up to time τ^*, processes $q_1', \ldots, q_{i_1}'$ are scheduled in a round robin fashion, and each other process in $C_2 \setminus C_1$ does not take any step. When process $p' \in C_1$ is scheduled, it takes just one step. If this step is a receive step, then p' receives every message designated to p' that has not been delivered yet. If this step is a query to the underlying failure detector X, then we set the step to take place at a time τ such that $H(p',\tau) = s_r^{(i_2)}$, where r is the rank of $\mathsf{id}(p')$ among the ids of the processes in C_2. Again, since $s_r^{(i_2)}$ is output infinitely often by X at process p', this is always possible. After time τ^*, we let every process take steps, where the schedule proceeds as described before except that it now includes the processes in $C_2 \setminus C_1$ in the round robin order.

Recall that, in execution $exec_1$, the set of correct processes is $\mathsf{correct}(\mathcal{F}_{C_1}) = C_1$. It follows from the definition of P that the emulation protocol ensures that, eventually, every query to P returns $\Pi \setminus C_1$. Hence, in $exec_1$, there exists $\nu \in \mathbb{N}$ such that, within the prefix of $exec_1$ consisting in its first ν steps, at least one query to P has returned $\Pi \setminus C_1$. We define τ^* as the least time until ν steps of $exec_2'$ have been taken.

Recall that there exists a strictly increasing map $m : [1, i_1] \to [1, i_2]$ such that $q_j' = q_{m(j)}$ and $s_j^{(i_1)} = s_{m(j)}^{(i_2)}$ for every $j, 1 \leq j \leq i_1$. Therefore, the prefixes consisting of the first ν steps of $exec_1$ and $exec_2'$ are indistinguishable from the perspective of the processes $q_1', \ldots, q_{i_1}'$. Indeed, in both prefixes, only processes $q_1', \ldots, q_{i_1}'$ take steps, each process $q_r' = q_{m(r)}$, $1 \leq r \leq i_1$ receives the same output from X (namely, $s_r^{i_1} = s_{m(r)}^{i_2}$), and $q_r' = q_{m(r)}$ takes the same steps, in the same order, in both prefixes. It follows that, in $exec_2'$, a query to the (emulated) failure detector P returns $\Pi \setminus C_1$ which contains the non-empty set $C_2 \setminus C_1$. This is violating the specification of the failure detector P, which requires that every set output by P contains no correct processes. This contradiction completes the proof of Theorem 3.1. □

4 Perfect failure detection with quasi-constant #bits per process

In this section, we show that there exists a failure detector emulating the perfect failure detector P that outputs values whose range depends on the size of the system, but increases extremely slowly with that size.

Theorem 4.1. *There exists a failure detector equivalent to P that, in any n-process system, outputs $\lceil \log \alpha(n) \rceil + 1$ bits at each process, where $\alpha : \mathbb{N} \to \mathbb{N}$ is a function that grows as least as slowly as the inverse Ackermann function.*

The rest of the section is dedicated to the proof of Theorem 4.1. A key ingredient to achieve failure detection with such a small amount of output values at each process is the notion of *distributed encoding of the integers*, recently introduced in [6]. Let Σ denote a finite or infinite alphabet of symbols. Recall from Section 3 that a word w over Σ is a finite sequence of symbols of Σ, and the length of w, i.e., the number of symbols in w, is denoted by $|w|$. Σ^* denotes the set of words of Σ, and a word $u = u_1, \ldots, u_k$ is a sub-word of word $v = v_1, \ldots, v_\ell$, denoted by $u =_* v$ if and only if $k \leq \ell$ and there exists a strictly increasing mapping $m : [1, k] \to [1, \ell]$ such that $u_i = v_{m(i)}$, for every i, $1 \leq i \leq k$. u is said to be a *strict sub-word* of v if $u =_* v$ and $u \neq v$ and $|u| < |v|$.

Definition 4.2 ([6])**.** *A* distributed encoding of the integers *is a pair (Σ, f) where Σ is a possibly infinite alphabet and $f : \Sigma^* \to \{\text{true}, \text{false}\}$ is a function such that, for every integer $n \geq 1$, there exists a word*

$w = w_1, \ldots, w_n \in \Sigma^n$ *satisfying* $f(w) =$ true *and* $f(w') =$ false *for every strict sub-word* $w' \in \Sigma^*$ *of* w*. The word* w *is called the* code *of* n*, denoted by* $\mathsf{code}(n)$*.*

A trivial example of a distributed encoding consists in setting $\Sigma = \mathbb{N}$, and encoding every integer n with the word $n, \ldots, n$ of length n. For any $s \in \mathbb{N}^*$, the function f returns true on input s if $s = |s|, \ldots, |s|$, and false otherwise. To encode the first n integers, this encoding uses words in an alphabet of n symbols, each symbols being encoded on $O(\log n)$ bits. We are interested in parsimonious distributed encodings of the integers, i.e., encodings that use fewer than $\log n$ bits to encode the first n integers. Given a distributed encoding $E = (\Sigma, f)$, and $n \geq 1$, let $\Sigma_n \subseteq \Sigma$ denote the set of all symbols used in the code of at least one integer in $[1, n]$. More precisely, for every $u \in \Sigma$, $u \in \Sigma_n$ if and only if u is a symbol appearing in $\mathsf{code}(k)$ for some $1 \leq k \leq n$. We get that E uses symbols encoded on $O(\log |\Sigma_n|)$ bits to encode the first n integers.

Theorem 4.3 ([6])**.** *There exists a distributed encoding of the integers* (Σ, f) *such that, for every integer* $n \geq 1$*,* $|\Sigma_n| \leq \alpha(n)$ *where* $\alpha : \mathbb{N} \to \mathbb{N}$ *grows as least as slowly as the inverse-Ackermann function.*

We now show how to use distributed encoding of the integer to encode a perfect failure detector. Let (Σ, f) be a distributed encoding of the integers. We define a failure detector, called *micro-perfect*, and denoted by μP, induced by (Σ, f). We then show that μP is equivalent to the perfect failure detector P. That is, for any distributed encoding (Σ, f), there is a protocol that emulates P in any environment whenever the failure detector μP induced by (Σ, f) is available, and, conversely, there is a protocol that emulates μP in any environment whenever the failure detector P is available. Combining these two results with Theorem 4.3 yields Theorem 4.1.

4.1 The failure detector μP

Let (Σ, f) be a distributed encoding of the integers. An instance of the failure detector μP is built on top of each such encoding. Given (Σ, f), the range of μP is Σ. We denote by w_i^τ the output $H(p_i, \tau)$ of a failure detector history for μP at time τ at process p_i. Let w^τ denotes the output sequence of the failure detector at time τ at the processes that have not crashed by time τ, ordered by processes IDs. More formally, $w^\tau = w_{j_1}^\tau, w_{j_2}^\tau, \ldots, w_{j_k}^\tau$ where $\{p_{j_1}, \ldots, p_{j_k}\} = \Pi \setminus \mathcal{F}(\tau)$, and $\mathsf{id}(p_{j_1}) < \ldots < \mathsf{id}(p_{j_k})$. For every failure pattern $\mathcal{F}$, a failure detector history H belongs to $\mu P(\mathcal{F})$ if and only if there exists $\ell \in [1, n]$ (recall that n is the number of processes) for which

- there exist $a_i \in \mathbb{N}$ for $i = 1, \ldots, \ell$, with $1 \leq a_\ell < a_{\ell-1} < \ldots < a_2 < a_1 \leq n$;
- there exist $\tau_i \in \mathbb{N}$ for $i = 0, \ldots, \ell$, with $0 = \tau_0 < \tau_1 < \ldots < \tau_\ell = +\infty$

such that, for every i with $1 \leq i \leq \ell$, and for every τ, τ' with $\tau_{i-1} \leq \tau, \tau' < \tau_i$, the four following conditions hold:

(C1) $w_j^\tau = w_j^{\tau'}$ for every p_j, i.e., the output of the failure detector does not change between τ_{i-1} and τ_i;

(C2) $w^\tau =_* \mathsf{code}(a_i)$, i.e., the word formed by the outputs of the failure detector at each process that has not crashed by time τ is a sub-word of the code of a_i;

(C3) $a_i \geq n - |\mathcal{F}(\tau)|$, i.e., a_i is an upper bound on the number of non-faulty processes during $[\tau_{i-1}, \tau_i)$;

(C4) $a_\ell = |\mathsf{correct}(\mathcal{F})|$, i.e., a_ℓ is the number of correct processes.

Let us consider some time τ, and let $k = |\Pi \setminus \mathcal{F}(\tau)|$ denote the number of processes that have not crashed by time τ. By concatenating the failure detector outputs of the non-crashed processes (ordered by process IDs), we obtain a word $w^\tau \in \Sigma^k$. The failure detector μP guarantees that this word is either the distributed code of the current number k of non-crashed processes, or a sub-word of the distributed code of some integer $k' > k$. Moreover, eventually, μP outputs the distributed code of the number of correct processes.

Protocol 1 Emulation of P using μP induced by (Σ, f). Code of Process p_i.

```
1: function P-QUERY()
2:     for all S ⊆ Π : p_i ∈ S do launch thread th_S computing CHECK(S) end for
3:     wait until ∃S : th_S terminates; stop all other threads th_S' for S' ≠ S
4:     return Π \ S
5: function CHECK(S)
6:     r ← 0; count ← 0
7:     repeat
8:         r ← r + 1 ; send query(S, r) to every p_j ∈ S
9:         wait until resp((S, r),w_j) has been received from every p_j ∈ S
10:        w ← w_{j_1}, ..., w_{j_s} where S = {p_{j_1}, ..., p_{j_s}} and id(p_{j_1}) < ... < id(p_{j_s})
11:        if f(w) = true then count ← count + 1 end if
12:    until count = n
13: when query(S, r) is received from p_j do
14:    w_i ← μP-QUERY(); send resp((S, r), w_i) to p_j
```

4.2 Failure μP can emulate the perfect failure detector P

Protocol 1 emulates the perfect failure detector P using μP, in any environment.

a) Description of the protocol. Each time a query to P occurs, the protocol strives to identify a set of processes that (1) contains every correct process and (2) does not contain the processes that have failed prior to the beginning of the query. Given such a set S, $\Pi \setminus S$ is a valid output for P (line 4), as it does not contain any correct process (Accuracy), and, if the query starts after every faulty process has failed, $\Pi \setminus S$ is exactly the set of faulty processes (Completeness).

When P-QUERY() is invoked by some process p_i, 2^{n-1} threads are launched (line 2), one for each subset of Π containing p_i. Thread th_S associated to set S consists in a **repeat** loop (lines 7–12), each iteration of which aiming at collecting the outputs of the underlying failure detector μP at the processes of S. Each iteration is identified by a round number r[3]. In iteration r, *query* messages are first sent to every process in S (line 8), and then p_i waits for a matching *response* message[4] from each process in S (line 9). Iteration r may never ends if some processes of S fail. Nevertheless, for at least one set S, namely the set of correct processes, every iteration of the associated thread th_S terminates.

Each of the *response* messages received by p_i contains the output w_j of μP at its sender p_j when the message is sent (line 14). Assuming that *response* have been received from each process $p_j \in S$, let w be the word obtained by concatenating the outputs of μP in these messages, ordered by process id (line 10). Recall that a valid history of failure detector μP can be divided into ℓ *epochs* $e_1 = [0, \tau_1), e_2 = [\tau_1, \tau_2), \ldots, e_\ell = [\tau_{\ell-1}, +\infty)$, for some $\ell \leq n$. In each epoch $e_k, 1 \leq k \leq \ell$, the output of μP at each process does not change and satisfy conditions **(C1)**–**(C4)** of the definition .

Let us assume that iteration r entirely fits within epoch e_k for some $k, 1 \leq k \leq \ell$. That is, every message *query* or *response* of that iteration is sent during e_k. Thus, by condition **(C2)**, w is a sub-word of $\mathsf{code}(a_k)$, i.e., the encoding of the integer a_k associated with epoch e_k by the distributed code (Σ, f). By using the function f, it can be determined whether $w = \mathsf{code}(a_k)$ or not. Indeed, for every proper sub-word w' of $\mathsf{code}(a_k)$, $f(w') =$ false and $f(\mathsf{code}(a_k)) =$ true (cf. Definition 4.2). Moreover, integer a_k is an upper bound on the number of alive processes in epoch e_k (cf. **(C3)**). Therefore, if $f(w) =$ true, then $w = \mathsf{code}(a_k)$, and, since $|w| = |S|$, we get $|S| = a_k$. Since all processes in S have not failed at the beginning of e_k (as each of them has sent a *response* in that interval), it follows that every process not in S has failed. Furthermore, if $k = \ell$, then a_ℓ is the number of correct processes (cf. **(C4)**), and thus in that case $\Pi \setminus S$ is the complete set of faulty processes. To summarize, if the word w collecting during iteration r satisfies $f(w) =$ true, and iteration r entirely fits within an epoch, then $\Pi \setminus S$ is a valid output of P.

Unfortunately, it may be the case that an iteration terminates while not fitting entirely within an epoch.

[3] Round numbers may be omitted, we keep them to simplify the proof of the protocol.

[4] *query* and *response* messages are implicitly tagged in order not to confuse messages sent during different invocations of P-QUERY().

The word w collected during that iteration may contain values output by the failure detector μP in distinct epochs. It is thus no longer guarantied that w is a sub-word of a valid code, and, from the fact that $f(w) = \text{true}$, it can no longer be concluded that $\Pi \setminus S$ is a valid output of P. Recall however that the ℓ epochs are consecutive, they span the whole time range (last epoch e_ℓ never ends), and there are at most n of them. Hence, if n iterations terminate, at least one of these iterations fits entirely in an epoch. In thread th_S, the variable *count* enumerate the number of iterations that terminate with an associated word w such that $f(w) = \text{true}$ (cf. line 11). When this counter reaches the value n, at least one successful iteration fitting entirely in an epoch has occurred, and $\Pi \setminus S$ can therefore be returned as a valid result of a query to P (cf. lines 3–4).

b) Proof of Correctness. To establish the correctness of Protocol 1, we fix an arbitrary distributed encoding of the integers (Σ, f), and μP is built on that encoding. We shall consider an infinite execution *exec* of Protocol 1, and let $\mathcal{F}$ denote the failure pattern in that execution. We assume that, in *exec*, P-QUERY() is called infinitely often at each correct process. Let $H \in \mu P(\mathcal{F})$ be the failure detector history associated with *exec*. Let also $\tau_0 < \ldots < \tau_\ell$ and $a_1 > \ldots > a_\ell$, for some $\ell, 1 \leq \ell \leq n$, be the sequence of times and integers, respectively, satisfying the four conditions in the definition of μP in H. We first show that if an invocation of CHECK(S) returns in Protocol 1, then S contains every process that has not yet crashed by the time the invocation returns.

Lemma 4.4. *Let $S \subseteq \Pi$ be non empty. Consider an invocation of* CHECK(S) *that begins at time τ_b and ends at time τ_e. Then there exists $\tau \in [\tau_b, \tau_e]$ such that $\Pi \setminus S = \mathcal{F}(\tau)$.*

Proof. Let us consider an invocation I of CHECK(S) by some process p that terminates. Let τ_b and τ_e be the times at which I starts and returns, respectively. Since I terminates, it consists in R iterations of the **repeat** loop (lines 7-12) for some integer R. We say that the rth iteration of that loop is *successful* if the sequence w of failure detector outputs collected during that iteration passes the test of line 11, i.e., if $f(w) = \text{true}$. The variable *count* is incremented each time an iteration is successful. Moreover, the invocation terminates when *count* reaches the value n. Let $r_1 < r_2 < \ldots r_n$ denote the successful iterations and let $\mathcal{R} = \{r_1, \ldots, r_n\}$.

For every integer $r \in \mathcal{R}$, let τ_b^r and τ_e^r be the times at which the rth iteration begins and ends, respectively. The output of μP remains the same along each interval $[0, \tau_1), [\tau_1, \tau_2), \ldots, [\tau_{\ell-1}, +\infty)$, where $\ell \leq n$. Hence, by the pigeonhole principle, at least one complete iteration occur in one single interval. That is, there exists $r \in \mathcal{R}$ and there exists i with $0 \leq i \leq \ell - 1$ such that

$$[\tau_b^r, \tau_e^r] \subseteq [\tau_i, \tau_{i+1})$$

where we recall that $\tau_0 = 0$ and $\tau_\ell = +\infty$.

For each process $p_j \in S$, the output of μP does not change in the interval $[\tau_i, \tau_{i+1})$. Let w_j be the output of the failure detector at p_j during that interval. It follows that the sequence w that p_j obtains in the rth iteration is $w = w_{j_1}, \ldots, w_{j_{|S|}}$ (cf. Line 10), where $S = \{p_{j_1}, \ldots, p_{j_{|S|}}\}$ and $\mathsf{id}(p_{j_1}) < \ldots < \mathsf{id}(p_{j_{|S}})$. By the second condition in the definition of μP, we have $w =_* \mathsf{code}(a_{i+1})$, where $\mathsf{code}(a_{i+1})$ is the encoding of a_{i+1} in the distributed encoding (Σ, f). Iteration r being successful, we have $f(w) = \text{true}$. Since no proper sub-word w' of $\mathsf{code}(a_{i+1})$ satisfies $f(w') = \text{true}$, we get that $w = \mathsf{code}(a_{i+1})$. Hence, $w \in \Sigma^{a_{i+1}}$, from which we conclude that $|S| = |w| = a_{i+1}$. By the third condition in the definition of μP, we get that

$$a_{i+1} = |S| \geq n - |\mathcal{F}(\tau)| \tag{1}$$

for every $\tau \in [\tau_i, \tau_{i+1})$.

Finally, for every $p_j \in S$, the process p whose invocation I of CHECK(S) terminates receives a message $\mathit{response}(\langle S, r\rangle, w_j)$ from p_j during iteration r. This message has been sent during the interval $[\tau_b^r, \tau_e^r]$. Hence, S is a subset of the processes that have not crashed by time τ, i.e.

$$\Pi \setminus \mathcal{F}(\tau) \supseteq S \tag{2}$$

for some time $\tau \in [\tau_b^r, \tau_e^r]$. By combining Eq. (1) and Eq. (2), we obtain that $S = \Pi \setminus \mathcal{F}(\tau)$. □

Lemma 4.5. *Let p_i be a correct process. Every invocation of P-QUERY at p_i terminates.*

Proof. Let p_i be a correct process. Let us assume, for the purpose of contradiction, that there is an invocation I of P-QUERY at p_i that does not terminate. Since p_i is correct, a thread th_{S^*} in which CHECK(S^*) is invoked is launched in I, where S^* denotes the set of correct processes. Since the communications are reliable, and since every process to which p_i sends a *query* message in I eventually replies with a *response* message, it follows that every iteration of the repeat-loop terminates. Moreover, it follows from the definition of μP that the following hold: There exists a time τ after which the output w_j of μP does not change, for every process $p_j \in S^*$, and the sequence $w = w_{j_1}, \ldots, w_{j_s}$ is the distributed encoding of the number s of correct processes, where $S^* = \{p_{j_1}, \ldots, p_{j_c}\}$ and $\mathsf{id}(p_{j1}) < \ldots < \mathsf{id}(p_{jc})$. Let us consider any iteration of the repeat-loop that starts after time τ. The sequence of failure detector outputs collected by p_i in that iteration is w. As w is the encoding of the integer $|S^*|$, it follows that $f(w) =$ true. Therefore, by instruction 11), the value of *count* is incremented. Thus, eventually, *count* $= n$ from which we conclude that CHECK(S^*) eventually terminates. □

Proposition 4.6. *For every distributed encoding (Σ, f), Protocol 1 implements the perfect failure detector P using the failure detector μP, in any environment.*

Proof. Let *exec* be an infinite execution of Protocol 1, and let p_i be a correct process. By Lemma 4.5, every invocation of P-QUERY at p_i terminates. Hence, for every process, and every query to P at that process, there is a matching response in *exec*. Let us consider any P-QUERY, together with its matching response, and assume that these events occur at time τ_1 and τ_2 respectively, at some process p_i. The output of this query is some set $T \subseteq \Pi$. Let $S = \Pi \setminus T$. Protocol 1 insures that there is an invocation of CHECK(S) that starts at time τ_b and ends at time τ_e with $[\tau_b, \tau_e] \subseteq [\tau_1, \tau_2]$. By Lemma 4.4, there exists some $\tau \in [\tau_1, \tau_2]$ such that $S = \Pi \setminus \mathcal{F}(\tau)$, i.e., $T = \mathcal{F}(\tau)$. It thus follows that the outputs of the invocations of P-QUERY satisfy the accuracy and completeness properties of P. □

4.3 Failure detector P can emulate the failure detector μP

Failure detectors P and μP are in fact equivalent. Protocol 2 emulates μP using the perfect failure detector P, in any environment.

Protocol 2 Emulation of μP induced by (Σ, f) using P. Code of Process p_i.

```
1: init alive ← {p_1, ..., p_n}; r ← 0
2: function μP-QUERY()
3:     repeat
4:         r ← r + 1; S ← P-QUERY(); alive ← alive \ S
5:         send query(r, alive) to all other processes
6:         repeat S ← P-QUERY()
7:         until response(r, a_j) has been received from every p_j ∈ Π \ S
8:         rec ← set of all received sets a_j
9:     until there exists a ⊆ Π such that rec = {a}
10:    k ← rank of id(p_i) in a; w_i ← kth symbol of code(|a|)
11:    return w_i
12: when query(r, a) is received from p_j do
13:    S ← P-QUERY(); alive ← (alive ∩ a) \ S; send response(r, alive) to p_j
```

a) Description of the protocol. Suppose that each process p_i is endowed with a local variable A_i whose content is a set of processes that satisfy the following conditions, where A_i^τ denotes the value of A_i at time τ :

1. at every time τ, the set A_i^τ contains every process that has not failed by time τ;
2. there exists times $\tau_0 < \tau_1 < \ldots < \tau_\ell = +\infty$ where $\ell \leq n$ such that, during each epoch $e_k = [\tau_k, \tau_{k+1})$, the value of each variable A_i does not change, and is equal to the same set S_k at each process;

3. after some finite time, for every process p_i, A_i is equal to the set of correct processes.

Let (Σ, f) denote a distributed encoding of the integer. Given the variable A_i, we simulate queries to the failure detector μP induced by (Σ, f) at process p_i. Process p_i reads the content of the variable A_i. The output of the query is then the kth symbol w_i of the code of $|A_i|$ in the distributed encoding (Σ, f), where k is the rank of $\mathsf{id}(p_i)$ in A_i. The symbol w_i is well defined since, by item 1, $p_i \in A_i$. Let us briefly check that this simple idea satisfies the properties of μP. By item 2, every query simulated by p_i during epoch e_k, for any $1 \leq k \leq \ell \leq n$ returns the same value (cf. Condition **(C1)** of Section 4). For conditions **(C2)**–**(C4)**, let us define $a_k = |S_k|$ as the integer associated with epoch e_k, where S_k is the common value of the variable $A_1, \ldots, A_k$ during epoch e_k. Let $w = w_{j_1}, \ldots, w_{j_m}$ where $w_{j_1}, \ldots, w_{j_m}$ are outputs of the simulated failure detector μP obtained by processes $p_{j_1}, \ldots, p_{j_m}$ in epoch e_k, with $\mathsf{id}(p_{j_1}) < \ldots < \mathsf{id}(p_{j_m})$. The word w is a sub-word of $\mathsf{code}(|S_k|) = \mathsf{code}(a_k)$ (cf. Condition **(C2)**). Moreover, by item 1, $|S_k|$ is an upper bound on the number of non-faulty processes during epoch e_k since S_k contains every process that has not failed at the beginning of that interval (cf. Condition **(C3)**). Finally, if $k = \ell$, i.e., if e_k is the last epoch, then $a_\ell = |S_\ell|$ is the number of correct processes according to item 3 (cf. Condition **(C4)**).

The goal of Protocol 2 is to provide each process p_i with sets of processes, as if those sets were obtained by reading variable A_i defined above. The complement of each of the sets returned by queries to the underlying failure detector P satisfies the two conditions 1 and 3 stated at the beginning of the protocol description, by the accuracy and completeness properties of P. However, in a valid history of P, it may be the case that the outputs of P at distinct processes are different, before the output has converged to the set of crashed processes.

Each process p_i maintains a local variable alive which contains a set of processes. Initially, the value of alive is the set Π of processes composing the system (cf. line 1). The value of alive at time τ is the set of processes that, to the knowledge of p_i, have not been suspected to have crashed by the underlying failure detector P. To that end, processes exchange the value of their local variable alive (cf. lines 5 and 13), and they periodically remove the processes appearing as results of query to P (cf. lines 4 and 13). Since no correct process is ever removed from a set alive, it follows from the completeness property of P that, eventually, the value of every variable alive is the set of correct process.

An invocation I of μP-QUERY() by process p_i consists in a loop (cf. lines 3–9) that terminates when p_i has received the same set $a \subseteq \Pi$ from each non-faulty processes. In each iteration of the loop, p_i sends a query message to every processes. Each matching response it receives carries the value of the local variable alive of the sender (cf. line 13). Note that the loop terminates since, after some time, the value of each variable alive is the set of correct processes. If the same set a is sent by every non-faulty process, then p_i behaves as if a was the value of A_i, and deduces a return value for the simulated query to μP (cf. lines 10–11). Indeed, consider another invocation I' of μP-QUERY() and let a' denote the common value of the alive variables collected in that invocation. At each process, the successive values of alive forms a decreasing sequence of sets in the inclusion order, $a \subseteq a'$ or $a' \subseteq a$. Moreover, if I' starts after I terminates, then a' is (non necessarily proper) subset of a. Hence, everything happens as if the successive values of the variables $A_1, \ldots, A_n$ form some sequence of sets $\Pi = a_1 \supset a_2 \supset \ldots \supset a_\ell = \mathsf{correct}$, and each invocation of μP-QUERY() is uniquely associated with one of these sets. See Lemma 4.10 and Proposition 4.11 for more details.

b) Proof of Correctness. To establish the correctness of Protocol 2, we fix an arbitrary distributed encoding of the integers (Σ, f), and we consider an arbitrary infinite execution *exec* of the protocol, in which every process keeps invoking μP-QUERY(). Let $\mathcal{F}$ be the failure pattern, and let H be the history of the failure detector P in *exec*. We denote by var_i^τ the value of a local variable var at process p_i at time τ. We first show that at every process p_i, the variable alive always includes the set of processes that have not yet crashed.

Lemma 4.7. *For every process p_i and every time τ, if $p_i \notin \mathcal{F}(\tau)$ then $\Pi \setminus \mathcal{F}(\tau) \subseteq \mathsf{alive}_i^\tau$.*

Proof. The proof is by induction on the time $\tau \in \mathbb{N}$. The lemma is true at the beginning of the execution (i.e., for $\tau = 0$) since the initial value of alive is Π at every process (cf. Instruction 1). So, let $\tau > 0$, and

let us assume that the lemma is true up to time $\tau - 1$. If no step is taken at time τ, or if the step taken at time τ does not modify of the content of any variable alive, then the lemma is still true at time τ because $\Pi \setminus \mathcal{F}(\tau) \subseteq \Pi \setminus \mathcal{F}(\tau - 1)$. Thus, let us assume that the step taken at time τ modifies the content of the variable alive at some process p_i, by Instruction 4 or 13. The new value alive_i^τ of that variable satisfies

$$\mathsf{alive}_i^\tau = \mathsf{alive}_i^{\tau-1} \setminus S_i^\tau$$

where S_i^τ is the output of P at time τ (by Instruction 4), or

$$\mathsf{alive}_i^\tau = (\mathsf{alive}_i^{\tau-1} \cap \mathsf{alive}_j^{\tau'}) \setminus S_i^\tau$$

for $j \neq i$ and $\tau' < \tau$ (by Instruction 13). Since a process id is never output by P before that process crashes, we get that

$$(\Pi \setminus \mathcal{F}(\tau)) \cap S_i^\tau = \emptyset.$$

Moreover, by the induction hypothesis, we have

$$\Pi \setminus \mathcal{F}(\tau) \subseteq \Pi \setminus \mathcal{F}(\tau') \subseteq \mathsf{alive}_j^{\tau'}$$

for every process p_j and every time $\tau' < \tau$. Therefore, we get $\Pi \setminus \mathcal{F}(\tau) \subseteq \mathsf{alive}_i^\tau$, as desired. □

Lemma 4.8. *There is a time τ after which the value of the variable alive at every correct process is equal to the set of correct processes.*

Proof. Let p_i be a correct process. It follows from the completeness and accuracy properties of P that, after some time τ, the output of P at p_i is always the set $S^* = \mathsf{faulty}(\mathcal{F})$ of faulty processes. Therefore, every iteration of the repeat-loop at p_i terminates. Indeed, for every messages $query(r, \cdot)$ sent by p_i to a correct process p_j, process p_i eventually receives a matching message $response(r, \cdot)$ from p_j. Hence p_i receives $response$ messages from every correct process, and, since, eventually, $S_i = S^*$ holds, the waiting condition of Instruction 6 is eventually satisfied. So, let us consider some time $\tau' \geq \tau$ at which p_i updates the content of its variable alive (this occurs at Instruction 4 or 13). Such a time τ' does exist because every iteration of the repeat-loop terminates.

Since $S_i^{\tau'} = S^*$, it follows from the code (instruction 4 or 13) that $\textsc{alive}_i^{\tau'} \cap S^* = \emptyset$. Therefore, by Lemma 4.7, $\mathsf{alive}_i^{\tau'} = \Pi \setminus \mathsf{faulty}(\mathcal{F}) = \mathsf{correct}(\mathcal{F})$. Thus, there exists some time after which the value of the variable alive is the set of all correct processes, at every correct process. □

Lemma 4.9. *Every invocation of μP-QUERY at every correct process terminates.*

Proof. Let p_i be a correct process. Let us consider an invocation I of μP-QUERY at p_i and let us assume, for the sake of contradiction that this invocation does not terminate. We have seen in the proof of Lemma 4.8 that every iteration of the repeat-loop at p_i terminates. Moreover, it follows from that Lemma that, after some time τ, $\mathsf{alive} = \mathsf{correct}(\mathcal{F})$ at every correct process where $\mathsf{correct}(\mathcal{F})$ denotes the set of all correct processes. Let us consider an iteration r of the repeat-loop by p_i that starts after every faulty process have crashed, and after time τ. In this iteration, every $response(r, a)$ message received by p_i has been sent by a correct process, and is such that $a = \mathsf{correct}(\mathcal{F})$. The halting condition of the repeat-loop (cf. Instruction 9) is thus satisfied. It follows that I terminates, yielding the desired contradiction. □

Let $\mathcal{I}$ be the set of invocations of μP-QUERY by any process in which the repeat-loop terminates. Observe that $\mathcal{I}$ includes every μP-QUERY invocation that terminates. Let us consider an invocation $I \in \mathcal{I}$, and let p_i be the process that performs that invocation. It follows from the Instruction 9 that, in the last iteration of the repeat-loop, each message $response()$ received by p_i carries the same set $a \subseteq \Pi$. That is, at process p_i, we have $\mathsf{rec} = \{a\}$ when the repeat-loop of invocation I terminates. We denote by $a(I)$ this set.

Lemma 4.10. *Let $I, I' \in \mathcal{I}$ be two invocations of μP-QUERY(). We have $a(I) \subseteq a(I')$ or $a(I') \subseteq a(I)$. Moreover, If I ends before I' begins then $a(I) \supseteq a(I')$.*

Proof. Let p_i and p_j be the processes at which invocations I and I' occur, respectively. Let p_k be an arbitrary correct process. In the last iteration of the repeat-loop in invocation I, process p_i receives messages $response(\cdot, a(I))$ from every processes in some set $T = \Pi \setminus S$, where S is the output of P at some time. As every set output by P contains no correct processes, p_i receives $response(\cdot, a(I))$ from p_k. Similarly, in invocation I', process p_j receives a message $response(\cdot, a(I'))$ from p_k in the last iteration of the repeat-loop of that invocation. Observe now that $a(I)$ and $a(I')$ are the values of the variable alive_k of p_k at some time τ and τ', respectively. Each time the value of alive is modified, the content of this variable is replaced by a subset of the previous set (cf. Instructions 4 and 13). Therefore $a(I) \supseteq a(I')$ or $a(I') \supseteq a(I)$, depending whether $response(\cdot, a(I))$ is sent before $response(\cdot, a(I'))$, or the other way around.

Finally, if I' starts after I has terminated, then $response(\cdot, a(I))$ is sent before $response(\cdot, a(I'))$ and thus $a(I) \supseteq a(I')$. □

Proposition 4.11. *For every distributed encoding (Σ, f), Protocol 2 implements μP using the perfect failure detector P, in any environment.*

Proof. By Lemma 4.9, for every correct process p_i, every invocation of μP at p_i terminates. It remains to show that the queries to μP can be linearized in such a way that all output values are compatible with some legal failure detector history in $\mu P(\mathcal{F})$. (Queries that do not belong to $\mathcal{I}$ do not terminate, and thus they do not need to be linearized). For the purpose of linearizing the queries that terminates, let $\leq$ be the following relation on $\mathcal{I}$. For every two invocations $I, I' \in \mathcal{I}$, we set

$$I \leq I' \iff \begin{cases} a(I) \subsetneq a(I') \text{ or} \\ a(I) = a(I') \text{ and } I \text{ starts not later than } I' \end{cases}$$

Let $I, I' \in \mathcal{I}$. By Lemma 4.10, it holds that $a(I) \subseteq a(I')$ or $a(I') \subseteq a(I)$. Hence, $I \leq I'$ or $I' \leq I$ for every two invocations I, I' of μP-QUERY. Moreover, the relation $\leq$ is antisymmetric (since at most one query starts at any given time), and transitive. Therefore $(\mathcal{I}, \leq)$ is a total order.

From Lemma 4.10, it also follows that $(\mathcal{I}, \leq)$ is compatible with "real-time". More precisely, for every two invocations $I, I' \in \mathcal{I}$, if I precedes I', i.e., if I' starts after I terminates, then $I \leq I'$. We can thus linearize every invocation in $\mathcal{I}$ according to the relation $\leq$. That is, for each invocation $I \in \mathcal{I}$, we choose a time τ_I between the time at which I starts and the time at which the repeat-loop is completed, such that, for every two invocations $I, I' \in \mathcal{I}$, we have:

$$I \leq I' \text{ and } I \neq I' \implies \tau_I < \tau_{I'} \ .$$

Let $A(\mathcal{I}) = \{a(I) : I \in \mathcal{I}\}$. Since $a(I) \subseteq \Pi$ for every query I, it follows that $A(\mathcal{I})$ is a finite collection of sets. Moreover, by Lemma 4.10, these sets are totally ordered by inclusion. So, let

$$A(\mathcal{I}) = \{A_1, \ldots, A_\ell\} \text{ with } A_1 \supset A_2 \supset \ldots \supset A_\ell \ .$$

Let $a_i = |A_i|$ for every $i \in [1, \ell]$. Let $\tau_0 = 0$, $\tau_\ell = +\infty$, and, for every $i \in [1, \ell)$, let us set τ_i as the first time at which a query I whose associated set is $a(I) = A_{i+1}$ is linearized. To complete the proof, we establish that the values returned by μP-QUERY are legitimate according to the specification of μP in Section 4. This specification is decomposed in four conditions **C1**, ..., **C4** related to the aforementioned values a_i's and τ_i's.

Claim. *Condition* **C1** *holds.*

We need to show that the output of the failure detector does not change between τ_{i-1} and τ_i. By definition of τ_{i-1} and τ_i, the set $a(I)$ associated with an invocation I linearized in the interval $[\tau_{i-1}, \tau_i)$ satisfies $a(I) = A_i$. Thus, A_i is the value of the local variable alive at some process(es), at some time $\tau' \leq \tau_{i-1}$. Thus, by Lemma 4.7, $\Pi \setminus \mathcal{F}(\tau') \subseteq A_i$. Let p_j be a process that has not failed at time $\tau \in [\tau_{i-1}, \tau_i)$. Since $p_j \in \Pi \setminus \mathcal{F}(\tau)$, p_j has not failed at time $\tau' \leq \tau$. Hence, $p_j \in A_i$. Therefore, by Instruction 10, every μP-QUERY by p_j linearized in that interval returns the same value, namely the kth symbol of the sequence $\mathsf{code}(|A_i|)$, where k is the rank of $\mathsf{id}(p_j)$ in the set of IDs of all the processes in A_i. Thus Condition **C1** holds.

In order to prove that the three remaining conditions hold, recall that w_j^τ denotes the value returned by μP-QUERY at process p_j at time τ, and let $w^\tau = w_{j_1}^\tau, \ldots, w_{j_k}^\tau$ where $\{p_{j_1}, \ldots, p_{j_k}\}$ are the processes that have not crashed at time τ and $\mathsf{id}(p_{j_1}) < \ldots < \mathsf{id}(p_{j_k})$.

Claim. Condition **C2** holds.

We need to show that $w^\tau =_* \mathsf{code}(a_i)$ for every $\tau \in [\tau_{i-1}, \tau_i)$, where $\mathsf{code}(a_i)$ is the distributed encoding of a_i. By **C1**, for each process p_j, every invocation of μP-QUERY at p_j linearized in $[\tau_{i-1}, \tau_i)$ returns a same value w_j^i, which the kth symbol in the sequence encoding $a_i = |A_i|$, where k is the rank of $\mathsf{id}(p_j)$ among the IDs of the processes in A_i. Let $\tau \in [\tau_{i-1}, \tau_i)$. By **C1**, the processes that are alive at time τ form a subset of A_i. Hence the sequence of values returned by $\mu P(\Sigma, f)$-QUERY at those processes is a sub-word of the sequence encoding $|A_i|$, i.e., $w^\tau =_* \mathsf{code}(a_i)$ as desired.

Claim. Condition **C3** holds.

We need to show that $a_i \geq n - |\mathcal{F}(\tau)|$ for every $i \in [1, \ell]$ and every $\tau \in [\tau_{i-1}, \tau_i)$. So, let $i \in [1, \ell]$ and $\tau \in [\tau_{i-1}, \tau_i)$. By definition, $a_i = |A_i|$, and $A_i = a(I)$ where I is an invocation of μP-QUERY that is linearized at time τ_{i-1}. As seen before, $a(I)$ is the value of some local variable alive at some time $\tau' \leq \tau_{i-1}$. It follows from lemma 4.7 that $\Pi \setminus \mathcal{F}(\tau') \subseteq A_i$. As $\Pi \setminus \mathcal{F}(\tau) \subseteq \Pi \setminus \mathcal{F}(\tau') \subseteq A_i$, we get that $a_i = |A_i| \geq |\Pi \setminus \mathcal{F}(\tau)|$, as desired.

Claim. Condition **C4** holds.

We need to show that $|a_\ell| = |\mathsf{correct}(\mathcal{F})|$. By Lemma 4.8, all the values of the local variables alive are eventually equal to the set of correct processes. Hence, eventually, for each invocation I of $\mu P(\Sigma, f)$-QUERY that terminates, $a(I) = \mathsf{correct}(\mathcal{F})$. Therefore, since processes keep invoking $\mu P(\Sigma, f)$-QUERY, and since, by Lemma 4.9, every invocation terminates at every correct process, we get that $A_\ell = \mathsf{correct}(\mathcal{F})$, from which we derive $a_\ell = |A_\ell| = |\mathsf{correct}(\mathcal{F})|$, as desired.

Thus, the four conditions **C1**, ..., **C4** are fulfilled, which completes the proof. □

4.4 Proof of Theorem 4.1.

By Theorem 4.3 there exists a distributed encoding of the integers (Σ, f) where $|\Sigma_n| \leq \alpha(n)$, for some $\alpha : \mathbb{N} \to \mathbb{N}$ growing at least as slowly as the inverse Ackermann function. That is, for any n, each symbol in the code of n is encoded on $\lceil \log \alpha(n) \rceil + 1$ bits. Consider failure detector μP induced by the distributed encoding (Σ, f). In an n-process system, any output of this failure detector is a symbol in Σ that is part of the code of some integer $n' \leq n$. Hence, the output of μP can be encoded on $\lceil \log \alpha(n) \rceil + 1$ bits at each process. Moreover it follows from the correctness of Protocols 1 and 2 that μP is equivalent to the perfect failure detector P. □

References

[1] A. R. Byron Cook, A. Podelski. Proving program termination. *Communications of the ACM*, 54(5):88–98, 2011.

[2] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996.

[3] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[4] C. Delporte-Gallet, H. Fauconnier, and S. Toueg. The minimum information about failures for solving non-local tasks in message-passing systems. *Distributed Computing*, 24(5):255–269, 2011.

[5] S. Dubois, R. Guerraoui, P. Kuznetsov, F. Petit, and P. Sens. The weakest failure detector for eventual consistency. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 375–384, 2015.

[6] P. Fraigniaud, S. Rajsbaum, and C. Travers. Minimizing the number of opinions for fault-tolerant distributed decision using well-quasi orderings. In *12th Latin American Theoretical Informatics Symposium (LATIN)*, LNCS #9644 pp. 497–508, Springer, 2016.

[7] P. Fraigniaud, S. Rajsbaum, C. Travers, P. Kuznetsov and F. Rieutord. Perfect failure detection with very few bits. Technical Report Hal#XXX, HAL, 2016. `https://XXXXX/XXXX`

[8] F. C. Freiling, R. Guerraoui, and P. Kuznetsov. The failure detector abstraction. *ACM Comput. Surv.*, 43(2):9, 2011.

[9] R. Guerraoui. Non-blocking atomic commit in asynchronous distributed systems with failure detectors. *Distributed Computing*, 15(1):17–25, 2002.

[10] C. Haase, S. Schmitz, and P. Schnoebelen:. The power of priority channel systems. *Logical Methods in Computer Science*, 10(4), 2014.

[11] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(2):326–336, 1952.

[12] P. Jayanti and S. Toueg. Every problem has a weakest failure detector. In *27th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 75–84, 2008.

[13] J. B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *J. Comb. Theory, Ser. A*, 13(3):297–305, 1972.

[14] M. Löb, and S. Wainer. Hierarchies of number theoretic functions. *I. Arch. MAth. Logic*, 13: 39–51, 1970.

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

[16] A. Mostéfaoui, S. Rajsbaum, M. Raynal, and C. Travers. The combined power of conditions and information on failures to solve asynchronous set agreement. *SIAM J. of Computing*, 38(4):1574–1601, 2008.

[17] A. Mostéfaoui, S. Rajsbaum, M. Raynal, and C. Travers. On the computability power and the robustness of set agreement-oriented failure detector classes. *Distributed Computing*, 21(3):201–222, 2008.

[18] A. Mostéfaoui and M. Raynal. Leader-based consensus. *Parallel Processing Letters*, 11(1):95–107, 2001.

[19] S. Schmitz and P. Schnoebelen. Algorithmic aspects of WQO theory. Technical Report Hal#00727025, HAL, 2013. `https://cel.archives-ouvertes.fr/cel-00727025`

[20] S. Schmitz and P. Schnoebelen. Multiply-Recursive Upper Bounds with Higman's Lemma In *38th International Colloquium in Automata, Languages and Programming (ICALP)* pages 441–452, LNCS#6756, Springer 2011.

[21] A. Turing. Checking a large routine. In *Conference on High Speed Automatic Calculating Machines*, pages 67–69, 1949.