View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Infoscience - École polytechnique fédérale de Lausanne

In Search of Lost Time

Bernadette Charron-Bost[1], Martin Hutle[2], and Josef Widder[3]
[1] CNRS / Ecole polytechnique, Palaiseau, France
[2] EPFL, Lausanne, Switzerland
[3] TU Wien, Vienna, Austria

Abstract

Dwork, Lynch, and Stockmeyer (1988) and Lamport (1998) showed that, in order to solve Consensus in a distributed system, it is *sufficient* that the system behaves well during a finite period of time. In sharp contrast, Chandra, Hadzilacos, and Toueg (1996) proved that a failure detector that, from some time on, provides "good" information *forever* is *necessary*. We explain that this apparent paradox is due to the two-layered structure of the failure detector model. This structure also has impacts on comparison relations between failure detectors. In particular, we make explicit why the classic relation is neither reflexive nor extends the natural history-wise inclusion. Altough not technically difficult, the point we make helps understanding existing models like the failure detector model.

Keywords: distributed computing – failure detectors – asynchronous system – Consensus

1 Introduction

The aim of the paper is to deepen the understanding of one of the most classical model in distributed computing, namely the failure detector model by Chandra and Toueg [2]. Roughly speaking, in the failure detector approach, a distributed system is represented by two layers: a layer that corresponds to the system itself — formally defined in terms of processes, automata, and communication links — that runs in a totally asynchronous mode, and a second layer consisting of the failure detector — defined with respect to time — accessible to the processes at any time.

From a modeling perspective, this two-layered approach radically changed from the classic approaches (e.g., [3, 4, 7]) where the timing behavior of a system is restricted in order to circumvent fundamental impossibility results [5]. Basically, the new idea in the failure detector model [2] was to consider distributed computations with unrestricted timing while the model is augmented with an oracle, expressed by a time-dependent layer that provides information from outside. In this paper, we explain how some characteristic features of this model actually originate from the interplay between the two layers, and the discrepancy between their timeless *vs*. time dependent definitions.

One of these features was proven in [1], namely, that Consensus cannot be solved without (implicit) *permanent* agreement on a leader from some time on. It seems in contradiction with earlier positive results by Dwork, Lynch, and Stockmeyer [4] and by Lamport [7] who showed that a sufficiently long *finite* "good" period makes Consensus solvable.

Another peculiarity of this bipartite model is the way failure detectors are compared: Chandra and Toueg [2] introduced a comparison relation which we show to be not reflexive. This is not just a formal problem that can be easily fixed. Actually, the problem again

originates from the interface between the system layer and the failure detector layer, and the lack of timing control by the failure detector on the asynchronous system layer.

1.1 Permanent *vs.* intermittent limitation of asynchrony

The central result in fault-tolerant distributed computing is that Consensus cannot be solved in an asynchronous system in the presence of crash failures [5]. Several approaches have been proposed to circumvent this impossibility result. In particular, Dwork, Lynch, and Stockmeyer [4] showed that Consensus can be solved if the asynchrony of the system is *intermittently* limited during a polynomial (in the size of the system and their timing parameters) amount of time. Similarly, Lamport proposed the *Paxos* algorithm [7] that solves Consensus if processes can elect a common leader for a sufficiently long finite period (the so-called "good periods"). These two positive results are very important as they both provide restrictions which are quite reasonable for implementing Consensus in practice.

Meanwhile, Chandra and Toueg [2] introduced the notion of failure detectors that are defined as external devices to the system: a failure detector is a distributed oracle which is supposed to provide information about failures, and so *augments* the underlying asynchronous system. Besides, Chandra, Hadzilacos, and Toueg [1] proved that a certain failure detector denoted Ω can be extracted from any failure detector that makes Consensus solvable. Roughly speaking, Ω provides the following information: "There is a time after which all the correct processes *always* trust the same correct process (the leader)." Contrary to the correctness conditions of the Consensus algorithms in [4, 7] recalled above, Ω is a very strong condition, clearly not achievable in real systems. In face of [4] and [7], it seems paradoxical that an eventually *forever* leader is *necessary* to solve Consensus.

The eventually forever requirement was discussed in [1]: it was stated that "in practice" it would suffice that some properties hold for *sufficiently long*, until an algorithm achieves its goal, and that

> [...] *in an asynchronous system it is not possible to quantify "sufficiently long", since even a single process step or a single message transmission is allowed to take an arbitrarily long amount of time.* [1]

Unfortunately, the argument of "the impossible quantification" of the length of good periods is fragmentary. Indeed, the point is to specify *in terms of what* a duration may be quantified. For instance, if one considers the number of messages in a causality chain and the number of steps taken by processes, one may quantify "sufficiently long" internally to an asynchronous system. The approach taken in [4, 7] can be interpreted in such a way.

In this paper, we make explicit that the origin of this necessity lies in the two-layered structure of the model. On the one hand, failure detectors are totally independent of the execution of the system, and failure detector histories are defined only with respect to time. On the other hand, the system is not "oracle-triggered": a process may query its failure detector module at any time, but cannot be incited by the failure detector or by some other oracle (e.g., a clock) to make a step. Hence, the system runs totally asynchronously: steps may occur at arbitrary times, and so may be arbitrarily delayed with respect to time. In order to be of any help to solve a problem, a failure detector must therefore be insensitive to such delays. In this paper, we formalize this intuition, and more generally we study the impact of a two-layered structure on the modeling. The point is not technically difficult, but we believe that a more thorough discussion on the basic properties of computational models for distributed systems is essential. In particular, it may help in better understanding the existing models such as the failure detector model.

To do that, we introduce for each failure detector $\mathcal{D}$, a failure detector $\widetilde{\mathcal{D}}$ which represents all the possible "samplings" of the histories of $\mathcal{D}$. More precisely, $\widetilde{\mathcal{D}}$ can be understood as the collection of all the possible time-contracted versions of $\mathcal{D}$, including $\mathcal{D}$ itself. Clearly, $\mathcal{D}$ is stronger than $\widetilde{\mathcal{D}}$ in the sense that $\mathcal{D}$ provides more precise information than $\widetilde{\mathcal{D}}$. The central point of our discussion is to show that given any (non real-time) problem P in asynchronous systems, $\widetilde{\mathcal{D}}$ is actually sufficient to solve P if P can be solved using $\mathcal{D}$. The basic reason lies in the fact that our time-contraction has no "visible" effect in an asynchronous system. In other words, the only information that can be exploited by an asynchronous system from a failure detector $\mathcal{D}$ is its time-contracted version $\widetilde{\mathcal{D}}$, i.e., *time is lost at the interface between the asynchronous system and failure detector modules*.

As a consequence, the weakest failure detector $\mathcal{W}_P$ to solve P, if it exists, is time-contraction insensitive. In particular, this holds for the failure detector Ω, which exactly captures the necessity of an *eventually forever* leader to solve Consensus.

1.2 Comparison relation between failure detectors

The relation $\succeq$ introduced in [2] to compare failure detectors is defined from an operational viewpoint: $\mathcal{D}' \succeq \mathcal{D}$ if there is an asynchronous distributed algorithm that transforms $\mathcal{D}'$ into $\mathcal{D}$. As explained above, timing information that may be given by the failure detector layer is definitely lost at the interface between the two layers. Thereby, the coupling between failure detectors and time — which by the way, may be very strong — cannot be handled by such an operationally defined relation. In other words, as soon as $\mathcal{D}$ is weaker than some other failure detector $\mathcal{D}'$ with respect to $\succeq$, $\mathcal{D}$ is in the restricted class of "time-free" failure detectors. In particular, non time-free failure detectors are not comparable to themselves, and $\succeq$ is not reflexive.

We shall explain why this shortcoming cannot easily be fixed in a two-layer model, and propose a new definition of a preorder (reflexive and transitive) for comparing failure detectors, which is quite different in essence: it compares failure detectors just with respect to their capabilities to solve problems. This relation extends both the original relation $\succeq$ and the inclusion relation between failure detectors.

2 Failure detectors, runs, and time

In this section we recall the definitions of the failure detector model to the extent necessary.[1] We denote by Π the finite set of processes which constitute the distributed system.

2.1 Failure detector layer

Central to the definition of failure detectors is a discrete global time $\mathcal{T} = \mathbb{N}$. A *failure pattern* is a function $F : \mathcal{T} \to 2^{\Pi}$. In the case of crash failures, $p \in F(t)$ means that p has crashed by time t, and so $F(t) \subseteq F(t+1)$. We say that p is correct in F if p never crashes, i.e., $p \in \mathit{correct}(F) = \bigcap_{t \in \mathcal{T}} \Pi - F(t)$, and we only consider failure patterns F such that $\mathit{correct}(F) \neq \emptyset$. An *environment* $\mathcal{E}$ is defined as a non-empty set of failure patterns.

A *history H with range $\mathcal{R}$*, where $\mathcal{R}$ is any non-empty set, is a function $H : \Pi \times \mathcal{T} \to \mathcal{R}$. A *failure detector* with range $\mathcal{R}$ is then defined as a function that maps each failure pattern $F \in \mathcal{E}$ to a non-empty set of histories with range $\mathcal{R}$.

Chandra and Toueg [2] introduced the notion of failure detector as a mapping from failure pattern to histories in order to capture the notion of an oracle that provides some

[1] For complete and formal definitions, the reader is referred to the original papers [2, 1].

hints about failures. In particular, a natural choice is $\mathcal{R} = 2^{\Pi}$, and $q \in H(p,t)$ for some history H of a failure detector $\mathcal{D}$ means that $\mathcal{D}$ tells p to suspect q to have crashed by time t. Alternatively, the failure detector Ω [1] has range $\mathcal{R} = \Pi$, and $q = H(p,t)$ encodes the fact that Ω tells p that q is correct. The key property of Ω is that from some time on, Ω provides the same information to all correct processes, and this information is correct.

The Ω property is invariant under time translation, allowing a large degree of freedom regarding time. This is not a general property shared by all failure detectors. Indeed, a failure detector may exhibit a strong relation to time: as an example, consider the history $H_{\mathcal{C}}$ with range $\mathcal{T}$ defined by

$$\forall p \in \Pi, \forall t \in \mathcal{T} : H_{\mathcal{C}}(p,t) = t,$$

and the *constant* failure detector $\mathcal{C}$:

$$\forall F \in \mathcal{E} : \mathcal{C}(F) = \{H_{\mathcal{C}}\}.$$

In sharp contrast to Ω, $\mathcal{C}$ gives no hint on failures but provides perfect information on time. Hence the term failure detector may be somewhat misleading as in the case of $\mathcal{C}$.

2.2 Asynchronous system layer

A distributed algorithm A over Π is a collection of deterministic automata $(A_p)_{p\in\Pi}$. Computation proceeds in *steps* of A. In each step, a process receives or not a message sent to it, may query its failure detector module that replies some value d to p, undergoes a state transition — depending on its current state, m, and d — and then sends a message to all other processes. We can thus identify this step with the triple (p, m, d).

A *schedule* of A is a sequence of steps taken by processes executing A; the i-th step of S is denoted by $S[i]$. The notion of the *applicability* of a step to a global state is then naturally[2] introduced and generalized to a schedule.

A *run of A using a failure detector $\mathcal{D}$* is a tuple $\langle F, H, I, S, T\rangle$, where F is a failure pattern, $H \in \mathcal{D}(F)$ is a history of $\mathcal{D}$, I is an initial global state of A, S is a schedule of A applicable to I, and T is a sequence of increasing values in $\mathcal{T}$ (such that the i-th element of T, $T[i]$, represents the time at which the step $S[i]$ occurs), and F, H, I, S, and T satisfy the following consistency conditions: $|S| = |T|$ and for any $i \leq |S|$, if $S[i] = (p, m, d)$, then $p \notin F(T[i])$ and $d = H(p, T[i])$. In addition, R must satisfy the following fairness properties: (1) each correct process takes an infinite number of steps in S, and (2) each message sent to a correct process is eventually received. The two layers of the model are thus connected solely *via* the condition which specifies that if process p takes a step at time t, then p changes its state depending among others on the value of $\mathcal{D}$ at time t, which is defined *externally* to the asynchronous system.

3 Comparing failure detectors

Numerous papers are devoted to determining a weakest failure detector to solve a given problem. All of them use the same comparison relation $\succeq$ which has been introduced in [2]. Roughly speaking, $\succeq$ is defined as follows: $\mathcal{D} \succeq \mathcal{D}'$ if there exists an algorithm A that "transforms" $\mathcal{D}$ into $\mathcal{D}'$. Thus, the relation $\succeq$ crosses the border between the two layers of the model in the sense that two failure detectors are compared through the asynchronous

[2] Ensuring consistency as requiring that a received message has been sent and has not been received before.

system layer. In this section, we show that the definition of $\succeq$ leads to some meaningless incomparability results, and we study how to fix the problem by extending the original relation $\succeq$.[3] We explain why trivial extensions are not satisfactory, and propose a new comparison relation which is a preorder, and whose definition does not resort anymore to the asynchronous system layer.

Let us recall what it means for an algorithm A to transform a failure detector $\mathcal{D}$ into another failure detector $\mathcal{D}'$: algorithm A uses $\mathcal{D}$ to maintain a variable out_p at every process p. This variable, reflected in the local state of A, emulates the output of $\mathcal{D}'$ at p. Let O_R be the history of all the out_p variables in run R, i.e., $O_R(p,t)$ is the value of out_p at time t in run R. Algorithm A *transforms* $\mathcal{D}$ *into* $\mathcal{D}'$ if for every run $R = \langle F, H, I, S, T\rangle$ of A using $\mathcal{D}$, $O_R \in \mathcal{D}'(F)$. If such an algorithm A exists, then $\mathcal{D} \succeq \mathcal{D}'$. The output history O_R highly depends on the time list T in R. More precisely, if $\mathcal{D}'$ is such that $\mathcal{D} \succeq \mathcal{D}'$ for some $\mathcal{D}$, then $\mathcal{D}'$ necessarily allows finite but unbounded stuttering: if process p does not take a step at time t in run R, then $O_R(p,t) = O_R(p,t-1)$. This condition on $\mathcal{D}'$ to be weaker than some failure detector $\mathcal{D}$ strongly restricts the graph of the relation $\succeq$, and in particular it prevents $\succeq$ to be reflexive, contrary to what can be inferred from the notation.

3.1 Extending inclusion

We could easily fix the problem of non-reflexivity by considering the reflexive closure of $\succeq$. Unfortunately, the resulting preorder is still too restrictive as we explain below. Consider the natural relation $\sqsubseteq$ between failure detectors:

$$\mathcal{D} \sqsubseteq \mathcal{D}' \Leftrightarrow \forall F \in \mathcal{E} : \mathcal{D}(F) \subseteq \mathcal{D}'(F).$$

In other words, $\mathcal{D}'$ provides additional possible histories compared to $\mathcal{D}$, and so is less accurate than $\mathcal{D}$. Hence, a natural requirement for a comparison relation over failure detectors is to be a preorder that extends $\sqsubseteq$. Unfortunately, this is the case neither for $\succeq$ nor for its reflexive closure. To see that, let us define the *instantaneous strong completeness* which ensures that every crashed process is immediately suspected by every correct process:

$$\forall F \in \mathcal{E},\ \forall H \in \mathcal{D}(F),\ \forall t \in \mathcal{T},\ \forall p \in \mathit{correct}(F), \forall q \in \Pi :\ q \in F(t) \Rightarrow q \in H(p,t),$$

and consider the two failure detectors $\mathcal{P}^+$ and $\mathcal{S}^+$ that both satisfy instantaneous strong completeness, but differ on their accuracy properties: $\mathcal{P}^+$ satisfies *strong accuracy* [2], i.e.,

$$\forall F \in \mathcal{E},\ \forall H \in \mathcal{D}(F),\ \forall t \in \mathcal{T},\ \forall p, q \in \Pi - F(t) : q \notin H(p,t),$$

whereas $\mathcal{S}^+$ satisfies only *weak accuracy* [2]

$$\forall F \in \mathcal{E},\ \forall H \in \mathcal{D}(F),\ \exists p \in \mathit{correct}(F),\ \forall t \in \mathcal{T},\ \forall q \in \Pi - F(t) : p \notin H(q,t).$$

Then one can easily show that:

Proposition 3.1. *$\mathcal{P}^+ \sqsubseteq \mathcal{S}^+$, but $\mathcal{P}^+ \not\succeq \mathcal{S}^+$.*

3.2 Time contraction

To fix the above problem, we can consider the relation $\succeq^*$ defined as the union of $\succeq$ and $\sqsubseteq$. Clearly, the relation $\succeq^*$ is a preorder, but it appears to be still too restrictive in the sense that it does not relate failure detectors that should be comparable from the standpoint of asynchronous systems, namely, failure detectors that only differ in their relation to time.

[3] In order to preserve all the weakest failure detector results, a new comparison relation must contain $\succeq$.

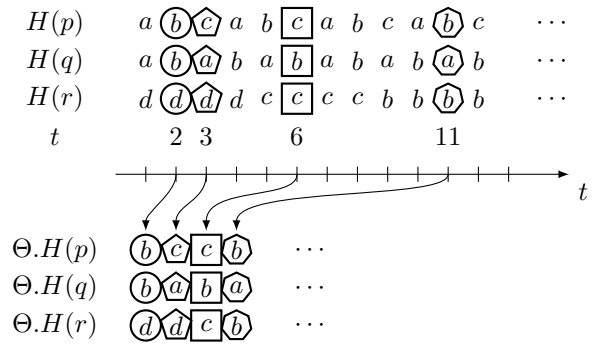

Figure 1: The history $\Theta.H$ with range $\{a, b, c, d\}$ and the sequence $\Theta = 2, 3, 6, 11, \cdots$

To see that, for any failure detector $\mathcal{D}$, we construct a new failure detector $\widetilde{\mathcal{D}}$ similar to $\mathcal{D}$ except the coupling to time that is weaker in $\widetilde{\mathcal{D}}$ than in $\mathcal{D}$. More precisely, each history of $\widetilde{\mathcal{D}}$ is obtained by sampling a history H of $\mathcal{D}$, i.e., taking at increasing times the values from H while "cutting out" the remaining values (see Fig. 1). In other words, $\widetilde{\mathcal{D}}$ is a time-contracted variant of $\mathcal{D}$, where the contraction is homogeneous between processes.

Let $\Theta = (\Theta_t)_{t\in\mathbb{N}}$ be an increasing sequence in $\mathcal{T}$, and $\mathbb{S}$ be the set of all such sequences. The contracted variants of a history H and a failure pattern F with respect to some $\Theta \in \mathbb{S}$ are defined by:

$$\Theta.H(p,t) = H(p,\Theta_t) \text{ and } \Theta.F(t) = F(\Theta_t).$$

Because failure patterns are non-decreasing functions of time, $\mathit{correct}(F) = \mathit{correct}(\Theta.F)$. In the following, we assume that $\mathcal{E}$ is such that each Θ-contraction $F \to \Theta.F$ is a surjection from $\mathcal{E}$ to $\mathcal{E}$.[4] Then, for every $\Theta \in \mathbb{S}$, we construct the failure detector $\Theta.\mathcal{D}$ which is obtained by the Θ-contraction of failure patterns and corresponding histories of $\mathcal{D}$:

$$\Theta.\mathcal{D}(F) = \left\{\Theta.H' :\ \exists F' \in \mathcal{E},\ F = \Theta.F' \ \wedge\ H' \in \mathcal{D}(F')\right\}.$$

We now define $\widetilde{\mathcal{D}}(F)$ as the union of *all* possible Θ-contractions of the histories in $\mathcal{D}(F)$:

$$\widetilde{\mathcal{D}}(F) = \bigcup_{\Theta\in\mathbb{S}} \Theta.\mathcal{D}(F)$$

Specializing $\Theta_t = t$, we obtain $\mathcal{D} \sqsubseteq \widetilde{\mathcal{D}}$, and so $\mathcal{D} \succeq^* \widetilde{\mathcal{D}}$. In general the converse inclusion does not hold, i.e., $\mathcal{D}$ and $\widetilde{\mathcal{D}}$ may be not equivalent with respect to $\succeq^*$.

Proposition 3.2. $\widetilde{\mathcal{C}} \not\succeq^* \mathcal{C}$

Proof. Let $\Theta \in \mathbb{S}$ be such that $\exists t \in \mathbb{N} :\ \Theta_t > t$. Thus, for any $F \in \mathcal{E}$ there is a history $H \in \widetilde{\mathcal{C}}(F)$ such that $\forall p \in \Pi :\ H(p,t) = \Theta_t > t$. Such a history $H \in \widetilde{\mathcal{C}}(F)$ is not in $\mathcal{C}(F)$, and thus $\widetilde{\mathcal{C}} \not\sqsubseteq \mathcal{C}$. Moreover, $\mathcal{C}$ does not allow stuttering, and so $\widetilde{\mathcal{C}} \not\succeq \mathcal{C}$. It follows that $\widetilde{\mathcal{C}} \not\succeq^* \mathcal{C}$. □

However, we now show that $\mathcal{D}$ and $\widetilde{\mathcal{D}}$ are always equivalent from the viewpoint of the asynchronous system layer in the sense that any asynchronous algorithm cannot distinguish $\mathcal{D}$ from $\widetilde{\mathcal{D}}$. In particular, we prove that $\mathcal{D}$ and $\widetilde{\mathcal{D}}$ have the same ability to solve (non real-time) problems in an asynchronous system.

[4] This assumption holds in the case of the classical environment where at most f processes may crash.

Let V_{in} and V_{out} be two non-empty sets such that $\perp \in V_{out}$ and $\perp \notin V_{in}$. An *initial configuration* is a function $C^0 : \Pi \to V_{in}$, and an *output sampling* is a function $\Sigma : \Pi \times \mathbb{N} \to V_{out}$.[5] A *problem* P is a predicate over triples (F, C^0, Σ), where $F \in \mathcal{E}$, C^0 is an initial configuration, and Σ is an output sampling. To remove any time dependency, we only consider problems such that

$$\forall F, F' \in \mathcal{E} \ : \ \mathit{correct}(F) = \mathit{correct}(F') \Rightarrow P(F, C^0, \Sigma) = P(F', C^0, \Sigma).$$

Let $A = (A_p)_{p \in \Pi}$ be an algorithm with the sets of states and of initial states denoted by $States_p$ and $States_p^0$, respectively. We say that A *solves* P if there exist two mappings $\sigma^0 : States_p^0 \to V_{in}$ and $\sigma : States_p \to V_{out}$ such that for any run $R = \langle F, H, I, S, T \rangle$ of A, P holds at $(F, \sigma^0(I), \sigma(I, S))$, where $\sigma^0(I)$ is the initial configuration corresponding to I by σ^0 and $\sigma(I, S)$ is the output sampling that naturally derives from I and S *via* σ.

If $\mathcal{D} \sqsubseteq \mathcal{D}'$ and A solves P using $\mathcal{D}'$, then A solves P using $\mathcal{D}$ since each run of an algorithm A using $\mathcal{D}$ is also a run of A using $\mathcal{D}'$. In particular, each run of A using $\mathcal{D}$ is also a run of A using $\widetilde{\mathcal{D}}$. Conversely, a run of A using $\widetilde{\mathcal{D}}$ may be not a run of A using $\mathcal{D}$. However, for each run R of A using $\widetilde{\mathcal{D}}$ there is a run R' of A using $\mathcal{D}$ with the same initial state and the same schedule as in R, and with equivalent failure patterns:

Proposition 3.3. *For every run $R = \langle F, H, I, S, T \rangle$ of an algorithm A using $\widetilde{\mathcal{D}}$ there is a run $R' = \langle F', H', I, S, T' \rangle$ of A using $\mathcal{D}$ such that $\mathit{correct}(F) = \mathit{correct}(F')$.*

Proof. For every history $H \in \widetilde{\mathcal{D}}(F)$ there is some $\Theta \in \mathbb{S}$ such that $H \in \Theta.\mathcal{D}(F)$. It follows that there exists some failure pattern $F' \in \mathcal{E}$ and some history $H' \in \mathcal{D}(F')$ such that $F = \Theta.F'$ and $H = \Theta.H'$. Let T' denote the increasing time values defined by

$$\forall i \in \mathbb{N} \ : \ T'[i] = \Theta_{T[i]}.$$

We are going to show that $R' = \langle F', H', I, S, T' \rangle$ is a run of A using $\mathcal{D}$. We know that $F' \in \mathcal{E}$, and I is an initial state of A. We denote by $S[i] = (p, m, d)$ the i-th step taken in S. Since R is a run of A, $d = H(p, T[i])$. By definition of H' and T', we have

$$d = H(p, T[i]) = \Theta.H'(p, T[i]) = H'(p, \Theta_{T[i]}) = H'(p, T'[i]).$$

Hence, $d = H'(p, T'[i])$. Furthermore, by definition of F' and T', we deduce that

$$F'(T'[i]) = F'(\Theta_{T[i]}) = F(T[i]).$$

It follows that if p takes the i-th step in S, then $p \notin F(T[i])$ as S is the schedule of run R, and thus $p \notin F'(T'[i])$, as needed. Finally, we know that each message sent in R to a process in $\mathit{correct}(F)$ is eventually received, and each correct process in F takes an infinite number of steps in R since R is a run. As F and F' are equivalent, R' also satisfies these two fairness properties. Consequently, R' is a run of A using $\mathcal{D}$. □

Theorem 3.4. *An algorithm A solves a problem P using $\mathcal{D}$ iff A solves P using $\widetilde{\mathcal{D}}$.*

[5] We use $\mathbb{N}$ instead of $\mathcal{T}$ to insist on the fact that $\Sigma(p, i)$ denotes the i-th output value and not the output value at time $i \in \mathcal{T}$.

3.3 Time regained

Hence, $\mathcal{D}$ and $\widetilde{\mathcal{D}}$ are equivalent in the asynchronous setting as they can be used to solve a problem with *the same algorithm*. However, Proposition 3.2 shows that there are failure detectors $\mathcal{D}$ such that $\widetilde{\mathcal{D}} \not\succeq^* \mathcal{D}$. We propose a comparison relation that keeps internal to the failure detector layer, and thus does not erase the possible timing information that failure detectors can contain. This new relation compares failure detectors just with respect to their capabilities to solve problems. Formally, $\mathcal{D} \succeq^s \mathcal{D}'$ if any problem solvable using $\mathcal{D}'$ is also solvable using $\mathcal{D}$. Clearly, the relation $\succeq^s$ extends $\sqsubseteq$, and $\succeq^s$ is reflexive. Moreover, $\succeq^s$ is transitive, and so it is a preorder. As claimed in [2], if $\mathcal{D} \succeq \mathcal{D}'$ then any problem solvable with $\mathcal{D}'$ is also solvable with $\mathcal{D}$, i.e., $\succeq^s$ extends the original relation $\succeq$. Finally, Theorem 3.4 shows that any failure detector $\mathcal{D}$ is equivalent to $\widetilde{\mathcal{D}}$ with respect to the relation $\succeq^s$.

4 On weakest failure detectors

We are now in position to explain the apparent paradox between the positive results of [4, 7] and the impossibility result in [1]. At some point, we will give only intuitive arguments, as we do not want to go into the details of the formal characterization of eventual forever properties in temporal logic.

Let $\mathcal{R}$ be a non-empty set and let Δ be the set of failure detectors with range $\mathcal{R}$. Given some problem P, we denote by $\Delta_P \subseteq \Delta$ the subset of failure detectors in Δ that can be used to solve P. Let $\gtrsim$ be any natural preorder to compare failure detectors in Δ, i.e., $\succeq^s$ extends $\gtrsim$ which in turn extends $\sqsubseteq$. Recall that $\mathcal{W}_P$ is *a weakest failure detector to solve P with respect to* $\gtrsim$ if $\mathcal{W}_P$ is a weakest element of $(\Delta_P, \gtrsim)$, i.e., (1) $\mathcal{W}_P \in \Delta_P$, and (2) for any $\mathcal{D}$ in Δ_P, $\mathcal{D} \gtrsim \mathcal{W}_P$. We now show that $\mathcal{W}_P$, if it exists, is necessarily a time-free failure detector.

Theorem 4.1. *If there exists a weakest failure detector $\mathcal{W}_P$ in Δ to solve P, then $\widetilde{\mathcal{W}}_P$ is also a weakest failure detector to solve P, and $\widetilde{\mathcal{W}}_P$ is equivalent to $\mathcal{W}_P$.*

Proof. Theorem 3.4 shows that $\widetilde{\mathcal{W}}_P \in \Delta_P$, and so $\widetilde{\mathcal{W}}_P \gtrsim \mathcal{W}_P$. Since $\mathcal{W}_P \sqsubseteq \widetilde{\mathcal{W}}_P$ and $\gtrsim$ extends $\sqsubseteq$, it follows that $\mathcal{W}_P \gtrsim \widetilde{\mathcal{W}}_P$. Hence, $\widetilde{\mathcal{W}}_P$ and $\mathcal{W}_P$ are equivalent. Moreover for any $\mathcal{D}$ in Δ_P, we have $\mathcal{D} \gtrsim \mathcal{W}_P$, and by transitivity $\mathcal{D} \gtrsim \widetilde{\mathcal{W}}_P$. This shows that $\widetilde{\mathcal{W}}_P$ is a weakest element in $(\Delta_P, \gtrsim)$. □

For a non-empty time interval τ, we define ϕ to be a condition on the collections of type

$$\Big(F_t, (D_{p,t})_{p\in\Pi}\Big)_{t\in\tau}$$

where for any $t \in \tau$ and any $p \in \Pi$, we have $F_t \subseteq \Pi$ and $D_{p,t} \in \mathcal{R}$. We say that a time interval τ is a *ϕ-good period for history H and failure pattern F* if ϕ is satisfied by

$$\Big(F(t), (H(p,t))_{p\in\Pi}\Big)_{t\in\tau}.$$

For instance, a ϕ-good period may express that "the same correct process is trusted by all processes".

Let P be a problem for which there is a weakest failure detector $\mathcal{W}_P$ to solve P that ensures ϕ-good periods in each of its histories and takes arbitrary values outside ϕ-good periods. Formally, for each failure pattern F and any history $H \in \mathcal{W}_P(F)$, (a) there is at least one ϕ-good period for H and F, and (b) any history H' with range $\mathcal{R}$, that coincides

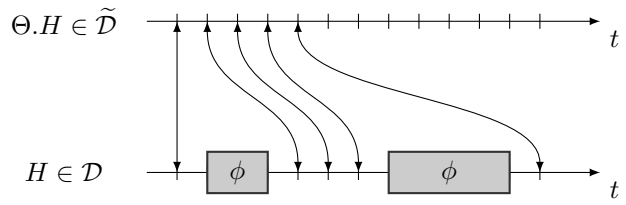

Figure 2: Finite good periods for H may lead to no good period for $\Theta.H$.

with H outside the ϕ-good periods, is also in $\mathcal{W}_P(F)$. Hence ϕ-good periods are necessary and sufficient in each history for solving P.

To seek a contradiction, suppose now that ϕ-good periods have only finite duration in some history $H_0 \in \mathcal{W}_P(F_0)$. Let $\mathcal{D}_0$ be the failure detector in Δ which coincides with $\mathcal{W}_P$ for any failure pattern $F \neq F_0$, and

$$\mathcal{D}_0(F_0) = \{H : \forall p \in \Pi, \forall t \in \mathcal{T}, H(p,t) \in \mathcal{R}\}.$$

Since $\widetilde{\mathcal{W}_P}$ consists of all samplings of the histories of $\mathcal{W}_P$, it follows that there is some history $H \in \widetilde{\mathcal{W}_P}(F_0)$ without any ϕ-good period (cf. Fig 2). Even, H may be totally arbitrary because of property (b) of $\mathcal{W}_P$. Hence, $\mathcal{D}_0 \sqsubseteq \widetilde{\mathcal{W}_P}$. Since $\gtrsim$ extends $\sqsubseteq$, it follows that $\mathcal{D}_0 \gtrsim \widetilde{\mathcal{W}_P}$. Moreover, Theorem 4.1 shows that $\widetilde{\mathcal{W}_P} \in \Delta_P$. As $\succeq^s$ extends $\gtrsim$, we derive that $\mathcal{D}_0 \in \Delta_P$. This shows that for F_0, a ϕ-good period is not necessary to solve P, a contradiction. Thereby, $\mathcal{W}_P$ ensures infinite ϕ-good periods in each of its histories.

In conclusion, ϕ-good periods must have infinite duration in order to be useful to solve a problem. In other words, ϕ must eventually hold forever, and this forever requirement stems solely from the modeling and the decomposition of the model into two layers.

5 Discussions

In this paper we pointed out the restrictive nature of the classical comparison relation between failure detectors, and we explained why in the failure detector model, Consensus cannot be solved without an eventually forever agreement on a process that will never crash, a non-realistic requirement in practice, whereas there are Consensus algorithms that run correctly in real systems. Both modeling problems originate from the interface between the failure detector layer and the asynchronous system layer, and from the lack of timing control by failure detectors on the asynchronous system: the time-contracted version of any failure detector $\mathcal{D}$ is equivalent to $\mathcal{D}$ from the viewpoint of the asynchronous system.

For comparing failure detectors, we proposed a new relation whose definition keeps internal to the failure detector layer, and thus preserves the possible timing information contained in failure detectors. On the contrary, there is no hope to weaken the properties on failure detectors needed for solving Consensus since the necessity of an eventually forever agreement is inherent to the nature of the model. This is the reason why we believe that the failure detector model, and more generally any model of asynchronous systems augmented with externally defined oracles, is not satisfactory as it leads to lower bound results that are invalidated by the experience of DLS or Paxos algorithms in real systems. In this respect, system models whose properties are defined internally, i.e., in terms of system transitions, appear more appropriate.

Jayanti and Toueg [6] recently established that every problem has a weakest failure detector with respect to yet another relation. While they mentioned that the classic relation in [2] is not reflexive, they did not discuss the paradox highlighted in here. Their new relation is reflexive and extends the inclusion relation $\sqsubseteq$. As our relation $\succeq^s$, this new relation also seems to extends the $\succeq$ relation of [2]. Unfortunately, Jayanti and Toueg [6] employed a new model, different from the classical failure detector model. Therefore it is delicate to establish a precise and formal comparison of these two different approaches of repairing the original relation. Note that the new model in [6] is still structured in several layers defined separately, and thus does not escape the paradox.

References

[1] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, June 1996.

[2] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[3] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.

[4] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

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

[6] Prasad Jayanti and Sam Toueg. Every problem has a weakest failure detector. In *Proceedings of the 27th ACM symposium on Principles of Distributed Computing (PODC)*, pages 75–84, 2008.

[7] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.