Indulgent Algorithms[*]
(Extended Abstract)

Rachid Guerraoui
Swiss Federal Institute of Technology,
CH-1015 Lausanne [†]

Abstract

Informally, an *indulgent* algorithm is a distributed algorithm that tolerates *unreliable* failure detection: the algorithm is *indulgent* towards its failure detector. This paper formally characterizes such algorithms and states some of their interesting features. We show that indulgent algorithms are inherently *safe* and *uniform*. We also state impossibility results for indulgent solutions to *divergent* problems like consensus, and *failure-sensitive* problems like non-blocking atomic commit and terminating reliable broadcast.

1 Introduction

Indulgent algorithms. The notion of *partial failures* is a fundamental characteristic of a distributed system: some of the processes might fail whereas others might keep executing their algorithm. A usual metric to evaluate the reliability of a system is its ability to mask partial failures. This ability typically relies on some failure detection mechanism that provides hints about which processes are correct and which are not. Distributed algorithms usually differ on the assumptions made about the reliability of that mechanism. Some algorithms assume failure detectors that accurately detect crashes. For example, the state machine replication algorithm of [15], the election algorithm of [12], and the non-blocking atomic commit algorithm of [16] assume that any correct process p_i accurately detects when any other process p_j has failed. Other algorithms do make weaker assumptions about failure detectors. For instance, none of the consensus algorithms of [2, 5, 9, 13], or the replication algorithms of [10, 11, 4], excludes the possibility of false failure detections. In a sense, those algorithms are *indulgent* (towards their failure detector).

Safety and uniformity. As we will show in the paper, some problems do not have indulgent solutions. Even when indulgent solutions exist, they are often complicated.

[*] "So when they continued asking him, he lifted up himself, and said unto them, He that is without sin among you, let him first cast a stone at her" - John 8:7.

[†]Contact e-mail: Rachid.Guerraoui@epfl.ch - regular paper submission.

The difficulty of devising an indulgent algorithm intuitively stems from the fact that processes should take some non-trivial "precautions" to cope with unreliable failure detection. Interestingly, and precisely because of those "precautions", indulgent algorithms turn out to have some "good" inherent characteristics: they are *safe* and *uniform*. The notions of *safety* and *uniformity* will be precised later in this paper, but to get an intuitive idea of their meaning, consider for instance the rotating-coordinator-based consensus [1] algorithm of [2], which we denote here by $\diamond\mathcal{S}$-cons. The algorithm assumes a $\diamond\mathcal{S}$ failure detector which ensures *strong completeness* (eventually every process that crashes is permanently suspected by every correct process), and *eventual weak accuracy* (eventually some correct process is not suspected by any correct process). Those properties do not prevent a failure detector from making an infinite number of false failure detections. Because the $\diamond\mathcal{S}$-cons algorithm needs to cope with those mistakes, it has the following interesting features:

- The $\diamond\mathcal{S}$-cons algorithm preserves safety even if neither of *strong completeness* nor *eventual weak accuracy* is satisfied. In particular, even if crashed processes are never suspected and correct processes are permanently suspected by all, (a) no two processes ever disagree on a decision and (b) no process ever decides on a value that was not proposed by some process. We say that the algorithm is *safe*.
- Although initially designed to solve consensus, $\diamond\mathcal{S}$-cons turns out to solve *uniform* consensus: a stronger variant of consensus where safety is preserved among all processes, whether they are correct or not.[2] We say that the algorithm is *uniform*.

This paper shows that *safety* and *uniformity* are inherent features of *indulgent* algorithms.

On the semantics of unreliability. Characterizing *indulgent* algorithms go through addressing a technical difficulty: defining what it means for a failure detector to be *unreliable*. One might be tempted for instance to consider as unreliable any failure detector that make mistakes, e.g., any failure detector that suspects a process to have crashed, even if that process is correct. An unreliability degree could then measure the number of mistakes that a failure detector makes [2]. This definition would however be counter-intuitive: a failure detector that never suspects any process (even a faulty one) would be *reliable*. Furthermore, the definition would only apply to failure detectors that output lists of suspects. The definition given in this paper more generally applies to any failure detector that outputs values that encode information about failures. Furthermore, it conveys the intuition that an unreliable failure detector is one that does never distinguish whether a process has crashed or is simply slow.

[1] In consensus [2], every process proposes an initial value 0 or 1, and must decide on a final value such that the three following conditions are satisfied: *agreement*, i.e., no two correct processes decide differently; *validity*, i.e., any value decided by a correct process is proposed by some process; and *termination*, every correct process eventually decides.

[2] In uniform consensus [7], beside the *termination* condition of consensus, the two following conditions need to be satisfied: *uniform agreement*, i.e., no two processes (correct or not) decide differently; and *uniform validity*, i.e., any value decided by a process (correct or not) is proposed by some process. [7] gives an algorithm that solves consensus but not solve uniform consensus. The algorithm assumes however a reliable failure detector.

We actually capture three variants of this intuition through three classes of failure detectors: the class of *weakly unreliable* failure detectors, denoted by $\triangle\mathcal{U}$, the class of *strongly unreliable* failure detectors, denoted by $\triangledown\mathcal{U}$, and the class of *completely unreliable* failure detectors, denoted by $\Box\mathcal{U}$. We consequently define three corresponding classes of indulgent algorithms: *weakly indulgent* that assume $\triangle\mathcal{U}$, *strongly indulgent* that assume $\triangledown\mathcal{U}$, and *completely indulgent* that assume $\Box\mathcal{U}$.

Contributions. This paper defines the notions of unreliable failure detectors and indulgent algorithms, points out examples of indulgent solutions to well-known agreement problems, and states the following results:

- *Safety.* Every strongly indulgent algorithm A is *safe*: informally, if A satisfies a safety property P with a failure detector $\mathcal{D}$, then A satisfies P even if $\mathcal{D}$ turns out to be completely unreliable.
- *Uniformity.* Every weakly indulgent algorithm A is *uniform*: informally, A cannot violate a property P without violating the *correct-restriction* [1] version of P. This result generalizes the result of [7] (any algorithm that solves consensus with $\diamond\mathcal{S}$ also solves uniform consensus).
- *Impossibility 1.* No weakly indulgent algorithm can satisfy any *globally-failure-sensitive* property (e.g., non-blocking atomic commit and terminating reliable broadcast) if one process can crash.
- *Impossibility 2.* No strongly indulgent algorithm can satisfy any *divergent* property (e.g., consensus if half of the processes can crash).[3]

We give our definitions and results in the asynchronous computation model [6] augmented with the failure detector abstraction [2, 3]. Basically, we assume a distributed system composed of a finite set Ω of $n > 1$ processes executing deterministic algorithms that use failure detectors. Processes can fail by crashing but every pair of processes is connected by a reliable communication channel. In this extended abstract, we do not detail the various definitions of the model [3]. Furthermore, we only informally introduce the new definitions that we had to add to the original model in order to state our results. For space limitation we also do not give any correctness proof. The complete model, definitions and proofs, are given in optional appendixes A, B, C, D and E.

Roadmap. Section 2 defines the notions of unreliable failure detectors and indulgent algorithms. Section 3 discusses the safety of indulgent algorithms whereas Section 4 discusses their uniformity and gives an impossibility result for globally-failure-sensitive properties. Section 5 gives a lower bound fault-tolerance result for divergent properties. Section 6 concludes the paper with some practical considerations.

[3]This result generalizes the lower bound of [2]: no algorithm can solve consensus using an eventually perfect failure detector if half of the processes can crash.

2 Unreliability and Indulgence

Intuitively, a failure detector is *unreliable* if it can never distinguish a faulty process from a correct one, and an *indulgent* algorithm is one that copes with unreliable failure detection. This section captures these intuitions more formally.

2.1 Unreliable failure detector classes

A *failure pattern* is a function F from Φ to 2^{Ω}, where $F(t)$ denotes the set of processes that have crashed through time t. Let F and F' be any two failure patterns. We say that F' *covers* F if $\forall t \in \Phi, F'(t) \subseteq F(t)$. An environment is a set of failure patterns and we assume here that every environment contains the failure-free pattern F_0 (where no process crashes) and at least one failure pattern where some process crashes. A *failure detector history* H with range G is a function H from $\Omega \times \Phi$ to G. For every process $p_i \in \Omega$, for every time $t \in \Phi$, $H(p_i, t)$ denotes the value of the failure detector module of process p_i at time t. A *failure detector* $\mathcal{D}$ is a function that maps each failure pattern F to a set $\mathcal{D}(F)$ of failure detector histories with range $G_{\mathcal{D}}$.

Definition (complete unreliability). *A failure detector $\mathcal{D}$ is completely unreliable if, for every pair of failure patterns F and F', $\mathcal{D}(F) = \mathcal{D}(F')$.*

We denote by $\Box\mathcal{U}$ the class of completely unreliable failure detectors. Consider for example failure detector $\mathcal{D}$ which always suspects all processes, at any time, any process, and in any failure pattern. Obviously, $\mathcal{D}$ is of class $\Box\mathcal{U}$. Similarly, failure detector $\mathcal{D}'$ which always outputs the empty set, at any time, any process, and in any failure pattern, is also of class $\Box\mathcal{U}$ ($\mathcal{D}'$ does never suspect any process).

Definition (strong unreliability). *A failure detector $\mathcal{D}$ is strongly unreliable if, for every pair of failure patterns F and F', for every history $H \in \mathcal{D}(F)$, for every time $t_i \in \Phi$, there is a failure detector history $H' \in \mathcal{D}(F')$ such that [$\forall t \leq t_i$, $\forall p_i \in \Omega$, $H'(p_i, t) = H(p_i, t)$].*

Informally, a failure detector is *strongly unreliable* if it *never* distinguishes a crashed process from one that is correct, and *vice et versa*. In other words, if a strongly unreliable failure detector $\mathcal{D}$ provides some information, at a time t and a process p, in a failure pattern F, $\mathcal{D}$ could have given the same information, at t and p, in any other failure pattern F'. We denote by $\bigtriangledown\mathcal{U}$ the class of strongly unreliable failure detectors. Consider $\Omega = \{p_1, p_2, p_3\}$ and let $\mathcal{D}$ be any failure detector of class $\bigtriangledown\mathcal{U}$. Let F_1 be any failure pattern where p_1 crashes while p_2 and p_3 are correct. At any time and any process, the information given by $\mathcal{D}$ in F_1 could have been given by $\mathcal{D}$ in any failure pattern F_2 where only p_2 crashes (or any failure pattern F_3 where p_1 and p_3 crash, etc.)

Definition (weak unreliability). *A failure detector $\mathcal{D}$ is weakly unreliable if, for every failure pattern F, for every history $H \in \mathcal{D}(F)$, for every failure pattern F' that covers F, for every time $t_i \in \Phi$, there is a failure detector history $H' \in \mathcal{D}(F')$ such that [$\forall t \leq t_i$, $\forall p_i \in \Omega$, $H'(p_i, t) = H(p_i, t)$].*

Informally, a failure detector is *weakly unreliable* (we simply say *unreliable*) if it *never* distinguishes a crashed process from one that is correct. In other words, if a *weakly unreliable* failure detector $\mathcal{D}$ provides some information at a time t and a process p_i, in a failure pattern F where a process p_j is faulty, $\mathcal{D}$ could have given the same information, at t and p_i, in a failure pattern F' similar to F, except that p_j is correct in F'. Any suspicion by $\mathcal{D}$ of process p_j may actually turn out to be false, i.e., p_j might be correct. We denote by $\triangle\mathcal{U}$ the class (the set) of weakly unreliable failure detectors. We call elements of $\triangle\mathcal{U}$ simply *unreliable* failure detectors. Consider for instance the set of processes $\Omega = \{p_1, p_2, p_3\}$ and let $\mathcal{D}$ be any failure detector of class $\triangle\mathcal{U}$. Let F_1 be any failure pattern where both p_1 and p_2 crash whereas p_3 is correct. At any time and any process, the information output by $\mathcal{D}$ in F_1 could have been given by $\mathcal{D}$ in some failure pattern F_2 where only p_1 crashes (or some failure pattern F_3 where all processes are correct).

Let A be any algorithm using a failure detector $\mathcal{D}$. We say that A is *completely indulgent* if $\mathcal{D} \in \Box\mathcal{U}$, *strongly indulgent* if $\mathcal{D} \in \bigtriangledown\mathcal{U}$, and weakly indulgent if $\mathcal{D} \in \triangle\mathcal{U}$. Any completely (resp. strongly) unreliable failure detector is strongly (resp. weakly) unreliable, i.e., $\Box\mathcal{U} \subset \bigtriangledown\mathcal{U} \subset \triangle\mathcal{U}$ (see Figure 1).[4] Consequently, every completely (resp. strongly) indulgent algorithm is strongly (resp. weakly) indulgent. When there is no need to distinguish between them, we call such algorithms simply *indulgent* algorithms.

2.2 Relations between failure detector classes

The aim of this section is to point out examples of indulgent algorithms. Instead however of explicitly exhibiting such algorithms, we state some intersection relations between our failure detector classes and the classes defined by Chandra and Toueg in [2][5] (these relations are depicted in Figure 1). By doing so, we indirectly show that some algorithms that have been described in the literature are indulgent.

Four main failure detector classes were defined in [2]: each class characterized by a *completeness* and an *accuracy* property: (1) the class $\mathcal{P}$ (*perfect*) characterized by *strong completeness* (eventually every process that crashes is permanently suspected by every correct process) and *strong accuracy* (no process is suspected before it crashes); (2) $\Diamond\mathcal{P}$ (*eventually perfect*) characterized by *strong completeness* and *eventual strong accuracy* (eventually no faulty process is ever suspected); (3) $\mathcal{S}$ (*strong*) characterized by *strong completeness* and *weak accuracy* (some correct process is never suspected); and (4) $\Diamond\mathcal{S}$ (*eventually strong*) characterized by *strong completeness* and *eventual weak accuracy* (eventually some correct process is never suspected).

Proposition 2.1 ($\mathcal{S} \cap \triangle\mathcal{U} \neq \emptyset$) *A failure detector can be unreliable and strong.*

---

[4]There are obvious examples of weakly (resp. strongly) unreliable failure detectors that are not strongly (resp. completely) unreliable. Appendix B gives examples of such failure detectors.

[5]It is important to notice that those classes were all defined according to *desirable* properties: completeness measures the extent to which a failure detector suspects the crash of faulty processes while accuracy restricts the mistakes made about failure suspicions. In contrast, our failure detector classes are defined according to *undesirable* properties that capture the intuition of unreliable failure detection.

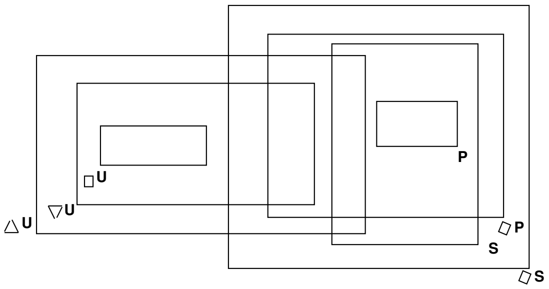

Figure 1: Intersection relations between failure detector classes

Proposition 2.2 **($\Diamond\mathcal{P} \cap \triangledown\mathcal{U} \neq \emptyset$)** *A failure detector can be strongly unreliable and eventually perfect.*

Proposition 2.3 **($\mathcal{S} \cap \triangledown\mathcal{U} = \emptyset$)** *No failure detector can be strong and strongly unreliable.*

Proposition 2.4 **($\mathcal{P} \cap \triangle\mathcal{U} = \emptyset$)** *No failure detector can be perfect and unreliable.*

Proposition 2.5 **($\Diamond\mathcal{S} \cap \Box\mathcal{U} = \emptyset$)** *No failure detector can be eventually strong and completely unreliable.*

A simple corollary of Proposition 2.1 (resp. Proposition 2.2) is that any algorithm that uses a strong (resp. eventually perfect) failure detector is indulgent (resp. strongly indulgent). For example, Chandra and Toueg described in [2] an algorithm $\mathcal{S}$-cons that solves consensus using any strong failure detector, and an algorithm $\Diamond\mathcal{S}$-cons that solves consensus using any eventually strong failure detector in environments with a majority of correct processes: $\Diamond\mathcal{S}$-cons is strongly indulgent and $\mathcal{S}$-cons is weakly indulgent. It was also shown in [2] that atomic broadcast and consensus are equivalent. This also means that atomic broadcast has a weakly indulgent solution in any environment, and a strongly indulgent solution in any environment with a majority of correct processes.

3 Safety

A property (or a problem) P is a set of runs (we use the term property as a synonymous to the term problem). We say that P *holds* in a run R if R is in P; P *holds* in a partial run R if P holds in any extension of R; and an algorithm A *satisfies* P if P holds in every run of A. In this paper, we restrict ourselves to properties that do not depend on failure detector values. These are properties P such that if two runs R and R' differ only in their failure detector history, then P cannot hold in R and not in R'.[6]

[6]We hence exclude properties of the form: "the failure detector is reliable". Such properties would pose a circularity problem since we use characteristics of failure detectors to derive results about algorithms that satisfy certain properties.

Definition (failure detector extension). We informally say that a failure detector $\mathcal{D}'$ *extends* a failure detector $\mathcal{D}$ if, at any time t and in any failure pattern F, the output given by $\mathcal{D}'$ could have be given by $\mathcal{D}$. (A formal definition is given in Appendix C.)

Informally, a *safety* property is one which states that "bad" things must not happen. More precisely, if the property does not hold in a run R, then R must have a partial run R' where P does not hold [14].[7]

Proposition 3.1 (safety) *Let A be any algorithm and P any safety property. If A satisfies P using a failure detector $\mathcal{D}$, then A satisfies P using any extension of $\mathcal{D}$.*

Proposition 3.2 (from strong to complete unreliability) *Every strongly unreliable failure detector $\mathcal{D}$ has a completely unreliable extension $\mathcal{D}'$.*

A simple corollary of propositions 3.1 and 3.2 is that if an algorithm A solves a problem P with a strongly unreliable failure detector $\mathcal{D}$, then A always preserves the safety aspects of P even if A is actually used with an extension of $\mathcal{D}$ that is completely unreliable. To illustrate this, consider a failure detector $\mathcal{D}$ that may (falsely) suspect any subset of processes until some arbitrary time t, and behaves perfectly after t: $\mathcal{D}$ is both strongly unreliable and eventually perfect. An algorithm A that uses $\mathcal{D}$ to solve a given problem P will never violate the safety aspects of P even if, instead of $\mathcal{D}$, A is actually used with a failure detector $\mathcal{D}'$ that keeps indefinitely behaving in an unreliable manner ($\mathcal{D}'$ is a completely unreliable extension of $\mathcal{D}$).

4 Failure Sensitivity

Some of the properties that have been defined in the literature are *failure-insensitive*: roughly speaking, the behavior of crashed processes does not impact the validity of these properties. Consensus is a typical example of a failure-insensitive property.

Many properties are however *failure-sensitive*, and their sensitivity to failures may come in different flavors. Some properties are failure-sensitive in the sense that they also restrict the behavior of faulty processes:[8] in *uniform consensus* for example, every faulty process should also respect agreement and validity. We say that such properties are *locally-failure-sensitive (lfs).* Other properties are *failure-sensitive* in a different sense. Consider a property, which we call here *atomic consensus*, defined with the termination and agreement condition of consensus, plus the following validity condition: 0 *can only be decided if some process crashes.* Atomic consensus does not restrict the behavior of crashed processes, but the very fact that some process

[7] NB. In contrast to a common belief, the agreement condition of consensus (as defined in the literature, e.g., [6]) does not define a safety property (in the sense of [14]). Agreement states that *no two correct processes should decide differently*: even if two processes has decided differently by some time t, there is still the hope that one of them will crash and validate agreement - this is rather a liveness property [14]. In order to conform to the intuition that agreement is indeed a safety property, one could rather phrase it as follows: *no two processes can decide differently unless at least one of them has crashed.* For agreement not to hold in a run R, there must be a partial run R' of R where two processes has decided differently and none of them has crashed: no extension of R' would satisfy agreement. The same observation applies to the validity condition of consensus.

[8] This notion corresponds to the notion of failure-sensitivity in [1].

p_i has crashed might globally impact the behavior of correct processes - even if p_i has initially crashed without executing any step. We say that *atomic consensus* is *globally-failure-sensitive (gfs)*. Atomic commitment [16] is *lfs* and *gfs*; terminating reliable broadcast [8] is *gfs* but not *lfs*; and uniform consensus is *gfs* but not *lfs*.

Definition (correct restriction). Given a property P, we informally define the *correct-restriction* of P as the property, denoted by $C(P)$, that is similar to P, except that $C(P)$ does not restrict the behavior of correct processes. (A formal definition is given in Appendix D.)

For instance, the correct-restriction of uniform consensus is consensus.[9]

Proposition 4.1 (uniformity) *Let A be any indulgent algorithm and P any safety property. If A satisfies $C(P)$ then A satisfies P.*

Definition (global failure sensitivity). We informally say that a property P is *globally-failure-sensitive* if the initial crash of some process restricts the behavior of correct processes. (A formal definition is given in Appendix D.)

Proposition 4.2 (impossibility) *No indulgent algorithm can satisy any globally-failure-sensitive property.*

A simple corollary of this result is that, unlike consensus and atomic broadcast, problems like non-blocking atomic commit and terminating reliable broadcast do not have (even weakly) indulgent solutions.

5 Divergence

Many properties in distributed systems involve agreement among a set of processes. These properties have a *divergent* flavour in the sense that the processes could potentially reach different (and contradictory) values, but in each run, these processes should agree on the same decision.

Definition (divergence). We informally say that a property is *divergent* if it can separately hold in two partial runs but not in the composition of these runs. (A formal definition is given in Appendix E.)

Consider $\Omega = \{p_1, p_2, p_3\}$, failure pattern F_1 where p_1 and p_2 initially crash whereas p_3 is correct, and failure pattern F_2 where p_3 initially crashes whereas p_1 and p_2 are correct. Consensus is typically divergent in any environment that contains F_1 and F_2. Consider configuration C where p_1 and p_2 initially propose 0 and p_3 proposes 1[10]. Starting from C, one could exhibit two partial runs of consensus, one where processes decide 1 and one where correct processes decode 0, such that consensus is violated in the composition of those runs. In contrast, reliable broadcast is not a divergent property [8].

[9] NB. If a property P is locally-failure-insensitive, then $C(P) = P$.
[10] C is a bivalent configuration in the sense of [6].

Proposition 5.1 (impossibility) *No strongly indulgent algorithm can satisfy any divergent property.*

Proposition 5.1 generalizes the lower bound fault-tolerance result of [2], which states that no algorithm can solve consensus using an eventually perfect failure detector if half of the processes can crash (consensus is divergent in any environment where half of the processes can crash).

6 Concluding Remarks

Distributed systems are rarely perfectly synchronous nor completely asynchronous. Process relative speeds and communication delays usually have upper timing bounds that can be determined, but there are sometimes unstability periods where those bounds are overrun. Failure detectors are typically implemented using time-outs and an application developer is left with a crucial dilemna: either to set up the time-outs with short values that ensure fast reaction to failures but increase the probability of false suspicions during unstability periods, or to choose large values that reduce the probability of false suspicions but introduce a slow fail-over. With a *strongly indulgent* algorithm, one can safely choose the first option or even consider dynamic time-outs. Despite false suspicions (during the unstability period of the system), safety is uniformly guaranteed among all processes. Liveness is eventually ensured when the system becomes stable again.[11] A *non-indulgent* algorithm might violate safety at the least false suspicion, and hence loses any chance to solve the problem even if the system becomes stable immediately after that suspicion.

Many practical distributed problems are *globally-failure-sensitive*, and as we stated in Proposition 4.2, such problems do not have indulgent solutions. Fortunately, most of those problems do often have a significant *globally-failure-insensitive* part. To illustrate this, consider non-blocking atomic commit. The agreement, validity and termination conditions of the problem define a sub-problem that is *globally-failure-insensitive* the problematic condition is non-triviality (i.e., *abort* cannot be decided if all processes vote *yes* and they are all *correct*) [7]. One could hence devise a strongly indulgent algorithm A_1 that solves the globally-failure-insensitive part of the problem, and uses this algorithm as a sub-algorithm withing the global algorithm A that solves the full problem (possibly assuming a reliable failure detector). If the failure detector makes mistakes (i.e., during an unstability period of the system), A might violate non-triviality and abort transactions that should have been otherwise committed. The developer could choose a large time-out value for the failure detector used by A (to reduce the probability of falsely aborting transactions) and a short time-out for the failure detector used by A_1: no matter what happens to the system, A would never violate agreement or validity. For example, the non-blocking atomic commit algorithm of [7] uses a strongly indulgent sub-algorithm to solve the agreement part of the problem. In contrast, the algorithm of [16] does not rely on any indulgent sub-algorithm and might violate agreement if the failure detector makes mistakes.

[11] "While there's life there's hope" - Cicero.

References

[1] R. Bazzi and G. Neiger. *Simulating crash failures with many faulty processors.* Distributed Algorithms (WDAG'92), Springer Verlag, LNCS 647, pages 166-184, 1992.

[2] T. Chandra and S. Toueg. *Unreliable Failure Detectors for Reliable Distributed Systems.* Journal of the ACM, 43(2), March 1996.

[3] T. Chandra, V. Hadzilacos and S. Toueg. *The Weakest Failure Detector for Solving Consensus.* Journal of the ACM, 43(4), July 1996.

[4] R. De Prisco, B. Lampson, and N. Lynch. *Revisiting the Paxos Algorithm.* Distributed Algorithms (WDAG'97), Springer Verlag, LNCS 1320, pages 111-125, September 1997.

[5] C. Dwork, N. Lynch, and L. Stockmeyer. *Consensus in the presence of partial synchrony.* Journal of the ACM, 35 (2), pages 288-323, April 1988.

[6] M. Fischer, N. Lynch, and M. Paterson. *Impossibility of Distributed Consensus with One Faulty Process.* Journal of the ACM, 32 (2), pages 374-382, April 1985.

[7] R. Guerraoui. *Revisiting the relationship between non-blocking atomic commitment and consensus.* Distributed Algorithms (WDAG'95), Springer Verlag, LNCS 972, pages 87-100, September 1995.

[8] V. Hadzilacos and S. Toueg. *Fault-Tolerant Broadcasts and Related Problems.* Cornell University, Technical Report (TR 94-1425), 1994.

[9] M. Hurfin and M. Raynal. *A Simple and Fast Consensus Algorithm in an Asynchronous System with a Weak Failure Detector.* Distributed Computing, 12 (4), 1999.

[10] L. Lamport. *The Part-Time Parliament.* ACM Transactions on Computer Systems, 16 (2), pages 133-169, May 1998.

[11] B. Liskov and B. Oki. *Viewstamped replication: A new primary copy method to support highly-available distributed systems.* ACM Symposium on Principles of Distributed Computing (PODC'88), pages 8-17, August 1988.

[12] L. Sabel and K. Marzullo. *Election Vs. Consensus in Asynchronous Systems.* Technical Report TR95-1488, Cornell Univ, 1995. Also, Technical Report CS95-411, UCSD, 1995.

[13] A. Schiper. *Early Consensus in an Asynchronous System with a Weak Failure Detector.* Distributed Computing, (10), pages 149-157, 1997.

[14] F.B. Schneider. *Decomposing Properties into Safety and Liveness.* Technical Report TR87-874, Cornell Univ, 1987.

[15] F.B. Schneider. *Replication Management using the State-Machine Approach.* In *Distributed Systems*, Sape Mullender (ed), Addison-Wesley, pages 169-197, 1993.

[16] D. Skeen. *NonBlocking Commit Protocols.* ACM SIGMOD International Conference on Management of Data, pages 133-142, 1981.

Appendix A: system model

We consider an asynchronous computation model augmented with the failure detector abstraction [2, 3]. The model is patterned after the FLP model [6]. Basically, we assume a distributed system composed of a finite set of n processes $\Omega = \{p_1, p_2, \ldots, p_n\}$ ($|\Omega| = n > 1$). There is no bound on communication delays or process relative speeds. A discrete global clock is assumed, and Φ, the range of the clock's ticks, is the set of natural numbers. The global clock is used for presentation simplicity and is not accessible to the processes.

Failure patterns

We assume that processes can only fail by *crashing*, i.e., they cannot behave *maliciously*. A process p_i is said to *crash at time t* if p_i does not perform any *action* after time t (the notion of *action* is recalled below). A *correct* process is a process that does not crash. Otherwise the process is said to be *faulty*. A *failure pattern* is a function F from Φ to 2^Ω, where $F(t)$ denotes the set of processes that have crashed through time t. Failures are permanent, i.e., no process *recovers* after a crash. In other words, $\forall t \leq t', F(t) \subseteq F(t')$. The set of correct processes in a failure pattern F is noted *correct*(F). Let F_1 and F_2 be any two failure patterns. We say that F_2 *covers* F_1 if for every time $t \in \Phi, F_2(t) \subseteq F_1(t)$. For instance, the failure-free pattern F_0 (where no process crashes) covers all failure patterns.

An *environment* E is a set of failure patterns. Environments describe the crashes that can occur in a system. We consider in this paper environments that contain the failure-free pattern F_0 and at least one failure pattern where some process crashes.

Failure detectors

Roughly speaking, a failure detector $\mathcal{D}$ is a distributed oracle which gives hints about failure patterns. Each process p_i has a local failure detector module of $\mathcal{D}$, denoted by $\mathcal{D}_i$. Associated with each failure detector $\mathcal{D}$ is a range $G_\mathcal{D}$ of values output by the failure detector.[12] A *failure detector history* H with range G is a function H from $\Omega \times \Phi$ to G. For every process $p_i \in \Omega$, for every time $t \in \Phi$, $H(p_i, t)$ denotes the value of the failure detector module of process p_i at time t, i.e., $H(p_i, t)$ denotes the value output by $\mathcal{D}_i$ at time t. A *failure detector* $\mathcal{D}$ is a function that maps each failure pattern F to a set of failure detector histories with range $G_\mathcal{D}$. $\mathcal{D}(F)$ denotes the set of possible failure detector histories permitted for the failure pattern F, i.e., each history represents a possible behavior of $\mathcal{D}$ for the failure pattern F.

Algorithms

An *algorithm* is a collection A of n deterministic automata A_i (one per process p_i). Computation proceeds in steps of the algorithm. In each step of an algorithm A, a process p_i atomically performs the following three actions: (1) p_i receives a message from some process q, or a "null" message λ; (2) p_i queries and receives a value d

[12] Unlike in [3], we denote a range by G, instead of R, in order to avoid confusions between runs and failure detector ranges.

from its failure detector module (d is said to be *seen* by p_i); (3) p_i changes its state and sends a message (possibly null) to some process. This third action is performed according to (a) the automaton A_i, (b) the state of p_i at the beginning of the step, (c) the message received in action 1, and (d) the value d seen by p_i in action 2. The message received by a process is chosen non-deterministically among the messages in the message buffer destined to p_i , and the null message λ. A *configuration* is a pair (I, M) where I is a function mapping each process p_i to its local state, and M is a set of messages currently in the message buffer. A configuration (I, M) is an initial configuration if $M = \emptyset$ (no message is initially in the buffer): in this case, the states to which I maps the processes are called *initial states*. A *step* of an algorithm A is a tuple $e = (p_i, m, d, A)$, uniquely defined by the algorithm A, the identity of the process p_i that takes the step, the message m received by p_i , and the failure detector value d seen by p_i during the step. A step $e = (p_i, m, d, A)$ is *applicable to a configuration* (I, M) if and only if $m \in M \cup_i \{\lambda\}$. The *unique* configuration that results from applying e to configuration $C = (I, M)$ is noted $e(C)$.

Schedules and runs

A *schedule* of an algorithm A is a (possibly infinite) sequence $S = S[1]; S[2]; \ldots S[k]; \ldots$ of steps of A. A schedule S is applicable to a configuration C if (1) S is the empty schedule, or (2) $S[1]$ is applicable to C, $S[2]$ is applicable to $S[1](C)$ (the configuration obtained from applying $S[1]$ to C), etc. Given any schedule S, we denote by $P(S)$ the set of the processes that take at least one step in S.

A *partial run* (resp. a *run*) is a tuple $R =< F, H, C, S, T >$ where, F is a failure pattern, H is a failure detector history, C is an initial configuration, T is a finite (resp. infinite) sequence of increasing time values, and S is a finite (resp. infinite) schedule (of some algorithm).

Let $R_1 =< F_1, H_1, C_1, S_1, T_1 >$ be any partial run, and $R_2 =< F_2, H_2, C_2, S_2, T_2 >$ any run. We say that R_2 is an *extension* of R_1 if $C_1 = C_2$, $\forall t \leq T_1[|T_1|]$, $\forall p_i \in \Omega$: $F_2(t) \subseteq F_1(t)$ and $H_1(p_i, t) = H_2(p_i, t)$, and $\forall i, 1 \leq i \leq |T_1|$: $S_1[i] = S_2[i]$ and $T_1[i] = T_2[i]$. We also say that R_1 is a *partial run* of R_2.

Let A be any algorithm and $\mathcal{D}$ any failure detector. A *partial run of A using* $\mathcal{D}$, is a tuple $R =< F, H, C, S, T >$ where H is a failure detector history such that $H \in \mathcal{D}(F)$, C is an initial configuration of A, T is a finite sequence of increasing time values, and S is a finite schedule of A such that, (1) $|S| = |T|$, (2) S is applicable to C, and (3) for all $k \leq |S|$ where $S[k] = (p_i, m, d, A)$, we have $p_i \notin F(T[k])$ and $d = H(p_i, T[k])$.

A *partial run of A using* $\mathcal{D}$, is a tuple $R =< F, H, C, S, T >$ where H is a failure detector history and $H \in \mathcal{D}(F)$, C is an initial configuration of A, S is an infinite schedule of A, T is an infinite sequence of increasing time values, and in addition to the conditions above of a partial run ((1), (2) and (3)), the two following conditions are satisfied: (4) every correct process takes an infinite number of steps, and (5) every message sent to a correct process q is eventually received by q.

Properties

A *property* (or a *problem*) is a set of runs. For instance, consensus is the set of runs for which agreement, termination and validity are satisfied [2] . Each of those sub-properties itself defines a set of runs. We say that a property P *holds* in a run R if R is in P. We say that P *holds* in a partial run R if P holds in any extension of R. An algorithm A *satisfies* a property P if P holds in every run of A.

Failure-detector-insensitivity. We say that a property P is *failure-detector-insensitive* if whenever P holds for a run $R =< F, H, C, S, T >$, P holds for any run of the form $R' =< F, H', C, S, T >$. Informally, P does not depend on the values output by the failure detectors. In the paper, we consider only such properties.

Appendix B: relations between failure detector classes

Proposition 2.1 ($\mathcal{S} \cap \triangle\mathcal{U} \neq \emptyset$) *A failure detector can be unreliable and strong.*

Proof: To show this result, we exhibit a "typical" unreliable failure detector $\mathcal{D}_w$ that satisfies the strong completeness and weak accuracy properties of $\mathcal{S}$. Failure detector $\mathcal{D}_w$ has range 2^Ω and is defined as follows:

- For every failure pattern F, $\mathcal{D}_w(F) = \{H \mid \exists p_k \in \mathit{correct}(F), \forall p_i \in \Omega, \forall t \in \Phi, p_k \notin H(p_i,t)$ and $\exists t_0 \in \Phi, \forall p_i \in \Omega : \forall t \geq t_0, H(p_i,t) = F(t)\}$.

Roughly speaking, in every failure pattern F, $\mathcal{D}_w$ might suspect all but some correct process q_k until some time t_0, and after time t_0, $\mathcal{D}_w$ suspects exactly the crashed processes, i.e., after time t_0, $\mathcal{D}_w$ behaves like a perfect failure detector. It is obvious that $\mathcal{D}_w$ is a strong failure detector: it satisfies both strong completeness and weak accuracy.

We show below that $\mathcal{D}_w$ is indeed unreliable. Consider any time $t_i \in \Phi$, any failure pattern F, and any history $H \in \mathcal{D}_w(F)$. By the definition of $\mathcal{D}_w$, there is a process $q_k \in \mathit{correct}(F)$ that is never suspected in H, and a time t_0 after which all processes in $faulty(F)$ are permanently suspected by all processes and no correct process is ever suspected. Consider any failure pattern F' that covers F (i.e., such that $\forall t \in \Phi\ F'(t) \subseteq F(t)$). Consider the history H' such that $[\forall t \leq t_i,\ \forall p_i \in \Omega,$ $H'(p_i,t) = H(p_i,t)$ and $\forall t > t_0, H(p_i,t) = F'(t)]$. As process $q_k \in \mathit{correct}(F)$, and $\mathit{correct}(F) \subseteq \mathit{correct}(F')$, then $q_k \in \mathit{correct}(F')$. Process q_k is thus never suspected in H and never suspected in H'. Furthermore, there is a time t_i after which, in H', all processes in $faulty(F')$ are permanently suspected and no correct process is ever suspected. Consequently, $H' \in \mathcal{D}_w(F')$, which means that $\mathcal{D}_w$ is unreliable. □

Proposition 2.2 ($\diamond\mathcal{P} \cap \triangledown\mathcal{U} \neq \emptyset$) *A failure detector can be strongly unreliable and eventually perfect.*

Proof: To show this result, we exhibit a "typical" strongly unreliable failure detector $\mathcal{D}_s$ which satisfies the strong completeness and eventual strong accuracy properties of $\diamond\mathcal{P}$. Failure detector $\mathcal{D}_s$ has range 2^Ω and is defined as follows:

- *For every failure pattern F, $\mathcal{D}_s(F) = \{H \mid \exists t_0 \in \Phi, \forall p_i \in \Omega, \forall t > t_0, H(p_i, t) = F(t)\}$.*

Roughly speaking, in every failure pattern F, $\mathcal{D}_s$ might suspect all processes until some time t_0, and after time t_0, $\mathcal{D}_s$ suspects exactly the crashed processes (after time t_0, $\mathcal{D}_s$ behaves like a perfect failure detector). It is obvious to see that $\mathcal{D}_s$ satisfies strong completeness and eventual strong accuracy. We show below that $\mathcal{D}_s$ is indeed strongly unreliable. Consider any time $t_i \in \Phi$, any failure pattern F, and any history $H \in \mathcal{D}_s(F)$. By the definition of $\mathcal{D}_s$, there is a time t_0 after which all processes in $faulty(F)$ are permanently suspected and no correct process is ever suspected. Consider any failure pattern F'. Consider the history H' such that $[\forall t \leq t_i,\ \forall p_i \in \Omega,\ H'(p_i, t) = H(p_i, t)$ and $\forall t > t_0, H(p_i, t) = F'(t)]$. After time t_i, all processes in $faulty(F')$ are permanently suspected and no correct process is ever suspected. Consequently, $H' \in \mathcal{D}_s(F')$, which means that $\mathcal{D}_s$ is strongly unreliable. $\Box$

Proposition 2.3 ($\mathcal{S} \cap \bigtriangledown\mathcal{U} = \emptyset$) *No failure detector can be strong and strongly unreliable.*

PROOF: (By contradiction) Let $\mathcal{D}$ be any failure detector that is both strong and strongly unreliable. Let F be any failure pattern where process p_j initially crashes and all other processes are correct. Let H be any history in $\mathcal{D}(F)$. By the properties of a strong failure detector, process p_j is never suspected and there is a time t after which all processes, except p_j are suspected by p_j. Consider the failure pattern F' where all processes except process $p_i \neq p_j$ initially crash and time $t' = t + 1$. As $\mathcal{D}$ is strongly unreliable, then there is a history H' in $\mathcal{D}(F')$ such that, for every process $p_i, H(p_i, t') = H'(p_i, t')$. As a consequence, there is a time t' at which the only process that is correct in F' is suspected by some correct process: a contradiction with the weak accuracy property of a strong failure detector. $\Box$

Proposition 2.4 ($\mathcal{P} \cap \bigtriangleup\mathcal{U} = \emptyset$) *No failure detector can be perfect and unreliable.*

PROOF: (By contradiction) Let $\mathcal{D}$ be a failure detector that is both perfect and unreliable. Let F be any failure pattern where some process p_j initially crashes and all other processes are correct. Let H be any history in $\mathcal{D}(F)$. By the strong completeness property of a perfect failure detector, there is a time t after which all correct processes permanently suspect p_j. Let p_i be any of those processes. Consider the failure-free pattern F_0. Obviously, F_0 covers F (i.e., $\forall t \in \Phi\ F_0(t) = \emptyset \subset F(t)$). By the definition of a unreliable failure detector, there is a history $H_0 \in \mathcal{D}_0$ such that $[\forall t \leq t_i,\ \forall p_i \in \Omega,\ H'(p_i, t) = H(p_i, t)]$. Hence at time t and history H', p_i suspects p_j in F_0 which is a failure-free: in contradiction with the strong accuracy property of a perfect failure detector. $\Box$

Proposition 2.5 ($\Diamond\mathcal{S} \cap \Box\mathcal{U} = \emptyset$) *No failure detector can be eventually strong and completely unreliable.*

PROOF (SKETCH): An asynchronous system model augmented with a completely unreliable failure detector is equivalent to a pure asynchronous system model. Assume that some completely unreliable failure detector is eventually strong. Such failure detector would solve consensus [2] and hence would contradict the FLP [6] impossibility result about solving consensus in an asynchronous system if one process can crash. □

Appendix C: safety

Safety property. We say that a property P is a *safety* property if any R where P does not hold has a partial run where P does not hold [14].

Failure detector extension. Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be any two failure detectors. We say that $\mathcal{D}_2$ *extends* $\mathcal{D}_1$ if for every failure pattern F, for every history $H_2 \in \mathcal{D}_2(F)$, for every time $t \in \Phi$, there is a failure detector history $H_1 \in \mathcal{D}_1(F)$, such that $[\forall t_i \leq t, \forall p_i \in \Omega, H_2(p,t) = H_1(p,t)]$. Informally, we say that $\mathcal{D}_2$ *extends* $\mathcal{D}_1$ if, at any time t and in any failure pattern F, the output given by $\mathcal{D}_2$ could have be given by $\mathcal{D}_1$.

Proposition 3.1 (safety) *Let A be any algorithm and P any safety property. If A satisfies P using a failure detector $\mathcal{D}$, then A satisfies P using any extension of $\mathcal{D}$.*

Proof: (By contradiction) Let A be any algorithm using a failure detector $\mathcal{D}$. Let $\mathcal{D}'$ be any extension of $\mathcal{D}$ with which A does not satisfy P. Since P is a safety property, then there is a partial run $R' =< F, H', C, S, T >$ of A using $\mathcal{D}'$ such that P does not hold in R. Since $\mathcal{D}'$ extends $\mathcal{D}$, then there is a failure detector history $H \in \mathcal{D}(F)$, such that $[\forall t_i \leq T[|T|], \forall p \in \Omega, H'(p_i,t) = H(p_i,t)]$. Partial run $R =< F, H, C, S, T >$ is also a partial run of A because (1) $|S| = |T|$, (2) S is applicable to C, and (3) for all $k \leq |S|$ where $S[k] = (p_i, m, d, A)$, we have $p_i \notin F(T[k])$ and $d = H(p_i, T[k])$. Let R'' be any run of A that extends R. Since R'' is an extension of R, then R'' is also an extension of R'. R'' is a run of A and P does not hold in R'': a contradiction. □

Proposition 3.2 (from strong to complete unreliability) *Every strongly unreliable failure detector $\mathcal{D}$ has a completely unreliable extension $\mathcal{D}'$.*

Proof: (By contradiction) Let $\mathcal{D}$ be any strongly unreliable failure detector. We construct a failure detector $\mathcal{D}'$ that (1) extends $\mathcal{D}$, and (2) is completely unreliable. Failure detector $\mathcal{D}'$ has the same range as $\mathcal{D}$, i.e., $G_{\mathcal{D}} = G_{\mathcal{D}'}$, and for every failure pattern F, $\mathcal{D}'(F) = \{H | \exists F', H \in \mathcal{D}(F')\}$.

We first show that $\mathcal{D}'$ is completely unreliable. Let F_1 and F_2 be any two failure patterns. Let H_1 be any history in $\mathcal{D}'(F_1)$. By the definition of $\mathcal{D}'$, there is a failure pattern F such that H_1 is in $\mathcal{D}(F)$. By the definition of $\mathcal{D}'$, this implies that H_1 is also in $\mathcal{D}'(F_2)$.

We show now that $\mathcal{D}'$ is an extension of $\mathcal{D}$. Let F' be any failure pattern and H' any history in $\mathcal{D}'(F')$. Consider any time $t \in \Phi$. By the definition of $\mathcal{D}'$, there is a failure pattern F such that H' is in $\mathcal{D}(F)$. Since $\mathcal{D}$ is strongly unreliable, then there is a history H in $\mathcal{D}(F')$ such that $[\forall t \leq t_i, \forall p_i \in \Omega, H'(p_i,t) = H(p_i,t)]$; which means that $\mathcal{D}'$ extends $\mathcal{D}$ □

Appendix D: failure-sensitivity

Correct-equivalence. Consider F any failure pattern and S any infinite sequence of steps. We denote by $correct(F,S)$ the restriction of S to correct processes in F. Let $R_1 =< F_1, H_1, C_1, S_1, T_1 >$ and $R_2 =< F_2, H_2, C_2, S_2, T_2 >$ be any two runs. We say that R_1 and R_2 are *correct-equivalent* if $correct(F_1, S_1) = correct(F_1, S_2)$.

Local-failure-sensitivity. We say that a property P is *locally-failure-sensitive* if for any two runs R_1 and R_2 that are correct-equivalent, P holds in R_1 iff P holds in R_2. Consensus is an example of a locally-failure-insensitive property: it does not restrict the behavior of correct processes. Consider two runs R_1 and R_2 of any consensus algorithm. Assume that the subsect Π of correct processes in both runs is the same. If consensus holds in a run R, and the correct processes of Π behave similarly in R and R', then no matter how faulty processes behave in R', consensus will indeed hold in R'. We say that a property is *locally-failure-insensitive* if it is not locally-failure-sensitive. Uniform consensus is for example a locally-failure-insensitive property.

Correct-restriction. Every property P has a locally-failure-insensitive part, which we call the *correct-restriction* of P. We define the *correct-restriction* of a property P, as the property denoted by $C(P)$ such that $C(P)$ does not hold in some run R iff P does not hold in every run that is correct-equivalent to R. Note that if a property P is locally-failure-insensitive, then $C(P) = P$. Consensus is for example the correct-restriction of consensus.

Proposition 4.1 (uniformity) *Let A be any indulgent algorithm and P any safety property. If A satisfies $C(P)$ then A satisfies P.*

We first introduce three lemmatas that are needed to prove the proposition.

Lemma 4.1 *Consider any property P and any run R with the failure-free pattern. If P holds in R then P holds in every run that is correct-equivalent to R.*

PROOF: Let $R_1 =< F_0, H_1, C_1, S_1, T_1 >$ and $R_2 =< F_0, H_2, C_2, S_2, T_2 >$ any two runs with the failure-free pattern F_0. If R_1 and R_2 are correct equivalent, then we have $S_1 = S_2$. Since we assume failure-detector-insensitive properties, then for any property P, P holds in R_1 iff P holds in R_2. □

Lemma 4.2 *Consider P any property and R any run with the failure-free pattern. $C(P)$ does not hold in R iff P does not hold in R.*

PROOF: If $C(P)$ does not hold in some run R then obviously P does not hold in R. Consider a run R with the failure-free pattern and assume that P does not hold in R. By Lemma 4.1 above, P does not hold in any run R' that is correct-equivalent to R. Hence, by the definition of the notion of correct-restriction, P does not hold in R. □

Lemma 4.3 *Let $R =< F, H, C, S, T >$ be any partial run of an algorithm A using an unreliable failure detector $\mathcal{D}$. For every failure pattern F' that covers F, there is a failure detector history $H' \in \mathcal{D}(F')$ such that $R' =< F', H', C, S, T >$ is also a partial run of A.*

PROOF: Let $R =< F, H, C, S, T >$ be any partial run of A. Consider the time $T[|T|]$. By the definition of a weakly unreliable failure detector, for every failure pattern F' that covers F (i.e., such that $\forall t \in \Phi, F'(t) \subseteq F(t)$), there is a failure detector history $H' \in \mathcal{D}(F')$ such that $[\forall t \leq T[|T|], \forall p \in \Omega, H'(p_i, t) = H(p_i, t)]$. We have $|S| = |T|$, S is applicable to C, and for all $i \leq |S|$ where $S[i] = (p, m, d, A)$, $p \notin F'(T[i])$ (as $correct(F) \subseteq correct(F')$) and $d = H'(p, T[i])$ (as $\forall t \leq T[|T|]$, $\forall p \in \Omega$, $H'(p_i, t) = H(p_i, t)$). Hence the partial run $R =< F', H', C, S, T >$ is also a partial run of A. □

PROOF OF PROPOSITION 4.1: (By contradiction) Let A be any algorithm using a weakly unreliable failure detector $\mathcal{D}$. Assume by contradiction that A satisfies $C(P)$ but does not satisfy P. Hence, there is a partial run of A using $\mathcal{D}$, $R =< F, H, C, S, T >$, such that P does not hold in R. Let F_0 be the failure-free pattern. By Lemma 4.3 above, $R_0 =< F_0, H, C, S, T >$ is also a partial run of A. Let $R'_0 =< F_0, H', C', S', T' >$ be any run of A that extends R_0. R'_0 is also an extension of R; which means that P does not hold in R'_0. Since F_0 is a failure-free failure pattern and P does not hold in R'_0, then by Lemma 4.2 above, $C(P)$ does not hold in R'_0: a contradiction. □

Global-failure-sensitivity. We say that a property P is *globally-failure-sensitive* if there is a configuration C, a failure pattern F, and a failure pattern F' that covers F, such that any run $R =< F, H, C, S, T >$ where P holds has a partial run $R' =< F', H', C, S, T >$ where P does not hold.

Informally, in a globally-correct-sensitive property, the very fact that some process has crashed might globally restrict the behavior of correct processes. Consider any environment E that has the failure-free pattern F_0 and at least one failure-pattern F where all processes are correct, except a process p_1 that initially crashes. Non-blocking atomic commit [16] is for instance globally-failure-sensitive in E. Consider the configuration C where all processes vote *yes*. Any run R, with configuration C and failure pattern F_1, where all correct processes decide *abort* satisfies atomic commit conditions. However, consider any run R', with C and F_0, where correct processes decide *abort*: atomic commit conditions do not hold in R' - the non-triviality condition of atomic commit is violated [7].

Proposition 4.2 (impossibility) *No indulgent algorithm can satisfy any globally-failure-sensitive property.*

PROOF: (By contradiction) Assume by contradiction that there is an algorithm A using an unreliable failure detector $\mathcal{D}$ that satisfies a global-failure-sensitive property P. Since P is *globally-correct-sensitive* then there is a configuration C, a failure pattern F, and a failure pattern F' that covers F, such that any run R with F and C where P holds has a partial run $R' =< F', H, C, S, T >$ where P does not hold. Consider time $T[|T|]$. Since $\mathcal{D}$ is unreliable and F' covers F, then there is a failure detector history $H' \in \mathcal{D}(F')$ such that: $[\forall t \leq T[|T|], \forall p_i \in \Omega, H(p_i, t) = H'(p_i, t)]$. Partial run $R'' =< F', H'', C, S, T >$ is also a partial run of A and any extension of R'' is an extension of R'. Since P does not hold in R' then it does not hold in R'': a contradiction. □

Appendix E: divergence

Run composition. Let F_1 and F_2 be any two failure patterns. We say that F_1 and F_2 are *disjoint* if $correct(F_1) \cap correct(F_2) = \emptyset$. Consider two partial runs $R_1 =< F_1, H_1, C_1, S_1, T_1 >$ and $R_2 =< F_2, H_2, C_2, S_2, T_2 >$. We say that R_2 *follows* R_1 if $T_2[1] > T_1[|T_1|]$.

Now, let F_1 and F_2 be any two disjoint failure patterns, and consider any two runs of the form $R_1 =< F_1, H_1, C_1, S_1, T_1 >$ and $R_2 =< F_2, H_2, C_2, S_2, T_2 >$ such that R_1 follows R_1. We define the composition of R_1 and R_2 as the partial run $R_1.R_2 =< F_0, H, C, S, T >$ such that: $C = C_1$, $\forall t \leq T_1[|T_1|]$, $\forall p_i \in \Omega$: $H(p_i, t) = H_1(p_i, t)$, $\forall t, T_1[|T_1|] < t \leq T_2[|T_2|]$, $\forall p_i \in \Omega$: $H(p_i, t) = H_2(p_i, t)$, $\forall i, 1 \leq i \leq |T_1|$: $S[i] = S_1[i]$ and $T_1[i] = T[i]$, and $\forall i, |T_1| < i \leq |T_1| + |T_2|$: $S[i] = S_2[i]$ and $T[i] = T_2[i]$.

Divergence. We say that a property P is *divergent* if there are two disjoint failure patterns F_1 and F_2, and there is a configuration C, such that any run $R_1 =< F_1, H_1, C, S_1, T_1 >$, where P holds, has a partial run $R_1' =< F_1', H_1', C, S_1', T_1' >$ such, for any run $R_2 =< F_2, H_2, C, S_2, T_2 >$ that follows R_1, where P holds, has a partial run R_2' such that P does not hold in $R_1'.R_2'$. We say that C is a *divergent configuration* of P for F_1 and F_2.

Consider $\Omega = \{p_1, p_2, p_3\}$ and the two following failure patterns: F_1 where p_1 and p_2 initially crash whereas p_3 is correct, and F_2 where p_3 initially crashes whereas p_1 and p_2 are correct. Consensus is typically divergent in any environment that contains F_1 and F_2. In fact, consensus has two divergent configurations for F_1 and F_2: C_1 where p_1 and p_2 initially propose 0 and p_3 proposes 1; and C_2 where p_1 and p_2 initially propose 1 and p_3 proposes 0. Starting from each of those configurations, one could exhibit two partial runs of consensus (one where processes decide 1 and one where correct processes decode 0) such that consensus is violated in the composition of those runs.

Proposition 5.1 (impossibility) *No strongly indulgent algorithm can satisfy any divergent property.*

PROOF: (By contradiction) Assume that there is an algorithm A using a strongly unreliable failure detector $\mathcal{D}$ that satisfies a divergent property in an environment with disjoint failure patterns F_1 and F_2. Since P is a divergent property, then P has a divergent initial configuration C for F_1 and F_2.

Consider failure pattern F_1, any failure detector history $H_1 \in \mathcal{D}(F_1)$, and the initial configuration C. Let R_1 be any run of A of the form $R_1 =< F_1, H_1, C, S_1, T_1 >$. Since A satisfies P, then P holds in R_1. Since P is divergent, then there is a partial run $R_1' =< F_1', H_1', C, S_1', T_1' >$ of R_1, such that for any run $R_2 =< F_2, H_2, C, S_2, T_2 > \in P$ such that $T_2[1] > T_1'[|T_1'|]$, there is a restriction R_2' of R_2 such that P does not hold in $R_1'.R_2'$.

Consider time $T_1'[|T_1'|]$ and failure pattern F_2. By the definition of a strongly unreliable failure detector, there is a failure detector history $H_2 \in \mathcal{D}(F_2)$ such that $[\forall t \leq T_1'[|T_1'|], \forall p_i \in \Omega, H_2(p_i, t) = H_1'(p_i, t)]$. Consider failure pattern F_2, failure detector history H_2, and configuration C. Let $R_2 =< F_2, H_2, C, S_2, T_2 >$ be any run of A starting at time $T_1'[|T_1'|] + 1$. Since A satisfies P then P does not hold in R_2. Since P is divergent, then there is a partial $R_2' =< F_2', H_2', C, S_2', T_2' >$ of R_2 such that P does not hold in $R_1'.R_2'$.

Consider time $T_2'[|T_2'|]$ and failure-free pattern F_0. By the definition of a strongly unreliable failure detector, there is a failure detector history $H_0 \in \mathcal{D}(F_0)$ such that $[\forall t \leq T_2'[|T_2'|], \forall p_i \in \Omega, H_0(p_i, t) = H_2'(p_i, t)]$.

Consider now the following partial run: $R_0 =< F_0, H_0, C, S_0 = S_1'.S_2', T_0 = T_1'.T_2' >$. We have: $|S_0| = |T_0|$, S_0 is applicable to C, and for all $i \leq |S|$ where $S_0[i] = (p_i, m, d, A)$, $p_i \notin F_0(T_0[i])$ (F_0 is the failure free pattern), and $d = H(p_i, T_0[i])$ because: $\forall p_i \in \Omega$, (1) $\forall t \in \Phi$ such that: $t \leq T_1'[|T_1'|]$, $H_0(p_i, t) = H_1'(p_i, t)$, and (2) $\forall t \in \Phi$ such that $T_1'[|T_1'|] < t \leq T_2'[|T_2'|]$, $H_0(p_i, t) = H_2'(p_i, t)$. R_0 is thus a partial run of A and any extension R of R_0 is also an extension of $R_1'.R_2'$. Let R be any extension of R_0. Since P is a divergent property, it does not hold in R: a contradiction. $\square$