The Weakest Failure Detector for Message Passing Set-Agreement

Carole Delporte-Gallet[1], Hugues Fauconnier[1], Rachid Guerraoui[2], and Andreas Tielmann[1,⋆]

[1] Laboratoire d'Informatique Algorithmique, Fondements et Applications (LIAFA), University Paris VII, France
[2] School of Computer and Communication Sciences, EPFL, Switzerland

Abstract. In the set-agreement problem, n processes seek to agree on at most $n-1$ different values. This paper determines the weakest failure detector to solve this problem in a message-passing system where processes may fail by crashing. This failure detector, called the *Loneliness* detector and denoted $\mathcal{L}$, outputs one of two values, *"true"* or *"false"* such that: (1) there is at least one process where $\mathcal{L}$ outputs always *"false"*, and (2) if only one process is correct, $\mathcal{L}$ eventually outputs *"true"* at this process.

Keywords: set-agreement, failure detectors.

1 Introduction

The set-agreement problem [1] has no deterministic solution in asynchronous systems where any number of processes can fail by crashing [2,3,4] and the remaining processes have no information about such failures. With failure detection however, the impossibility can be circumvented [5]. For instance, with a perfect failure detection mechanism that accurately detects crashes, it is trivial for the processes to reach agreement. A natural question is what failure information is necessary and sufficient to reach agreement. In the parlance of [6], this question can be precisely formulated using the notion of "weakest failure detector": In short, the weakest failure detector to solve a problem is one that (a) indeed solves the problem and (b) can be emulated by any failure detector that solves the problem. Property (a) conveys the sufficiency of the failure detector whereas property (b) conveys its necessity.

Several papers have been devoted to determine the weakest failure detector to solve the set-agreement problem in a distributed system where any number of processes can fail by crashing [7,8,9,10]. In particular, Zieliński proved recently that anti-Ω – a failure detector that outputs id's of processes such that the id of at least one correct process is output only finitely many times – is the weakest failure detector for set-agreement in a shared memory system [10]. The proof of

⋆ Work was supported by grants from Région Ile-de-France.

G. Taubenfeld (Ed.): DISC 2008, LNCS 5218, pp. 109–120, 2008.

the result is particularly involved and builds on earlier proof techniques from [6] and [8].

In the context of message passing however, the weakest failure detector for set-agreement has not been determined yet and one might have hoped to derive it somehow from anti-Ω. Indeed, Zieliński conjectured in [11] that failure detector Σ [12] – the weakest failure detector to build a shared memory in a message passing system – is both sufficient and necessary to implement set-agreement. This would mean that some common denominator of anti-Ω and Σ would constitute the weakest failure detector for set-agreement in message passing. Nevertheless, Delporte et al. recently disproved Zieliński's conjecture by showing that Σ is not necessary, albeit sufficient [13]. The question of the weakest failure detector to solve set-agreement in a message passing system remained thus open. The contribution of this paper is precisely to close the question.

We introduce the *Loneliness* failure detector, denoted $\mathcal{L}$, and we show that it is the weakest failure detector for set-agreement in a message passing system. Failure detector $\mathcal{L}$ outputs, whenever queried by a process, one of two values: *"true"* or *"false"* such that the following two properties are satisfied: (1) there is at least one process where the output is always *"false"*, and (2) if only one process is correct (does not crash), then the output at this process is eventually *"true"* forever. We first give an algorithm that solves set-agreement using $\mathcal{L}$. The particularity of the algorithm is its non-symmetric nature as it heavily exploits the total order on the identity of the processes. We then assume that there is an algorithm A that solves set-agreement (with some failure detector), and we show how to "extract" from A the output of $\mathcal{L}$. Our approach here is, on the one hand, different from the approach of [6] where each process locally simulates several runs of A and, on the other hand, different from the approach of [10], as well as [8], where the extraction relies on the asynchronous impossibility of a problem. In our case, the processes execute one instance of A, without knowing the automaton of A performed at each process. The processes obtain the output of $\mathcal{L}$ by "simply" intercepting communication between these automata. This leads to a very simple, almost trivial, extraction algorithm.

Our proof that $\mathcal{L}$ is the weakest in message passing is thus remarkably simple and this might be surprising compared to the rather involved proof of Zieliński [10] in shared memory systems. Somehow, we show that – contrary to a wide belief – results in message passing systems are sometimes easier to prove than in shared memory.

We prove that – not surprisingly – failure detector $\mathcal{L}$ is strictly stronger than anti-Ω, the weakest in a shared memory system. (Indeed a message passing system can be emulated by a shared memory system but the converse requires additional assumptions, e.g., a majority of correct processes [14].) Furthermore, we show that no failure detector that may behave arbitrarily for any finite amount of time is stronger than $\mathcal{L}$ (but nevertheless such failure detectors can be incomparable with $\mathcal{L}$). We also show that for $n > 2$, Σ is strictly stronger than $\mathcal{L}$, confirming the result of [13] that emulating a shared memory requires more information about failures than reaching agreement (Figure 1).

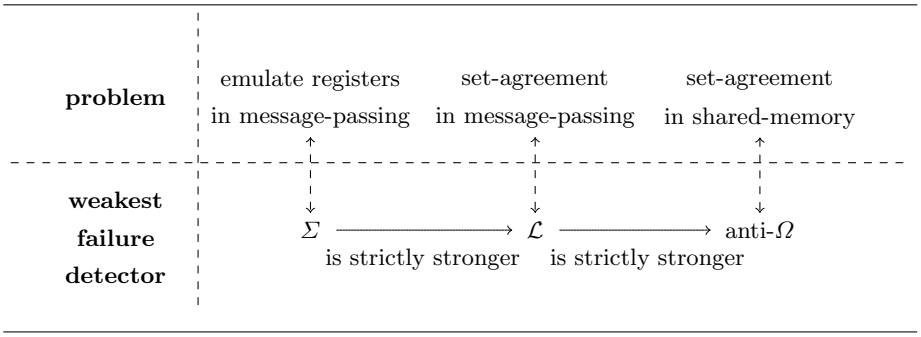

Fig. 1. Relations between failure detector classes

The rest of the paper is organized as follows. We first define our model in Section 2. Then we show that $\mathcal{L}$ is sufficient for set-agreement in Section 3 and that $\mathcal{L}$ is also necessary in Section 4. In Section 5, we show that $\mathcal{L}$ is strictly stronger than anti-Ω. And finally, in Section 6 we show that for $n > 2$, Σ is strictly stronger than $\mathcal{L}$.

2 Model and Definitions

2.1 Processes and Failure Detectors

The system model we consider is that of Chandra et al. [6] which we briefly recall here. We consider a set $\Pi = \{p_1, \ldots, p_n\}$ of $n \geq 2$ processes which communicate by message passing over a fully connected network with reliable links. Any number of processes may fail by prematurely halting, i.e. they crash. However, no process can otherwise deviate from its protocol. We assume a global clock $\mathcal{T}$ that is used to depict steps in an execution; the clock is not accessible to the processes.

A failure pattern is a function from time $\mathcal{T}$ to 2^{Π} that specifies for every time t which processes have crashed by time t. A process p_i that does not crash in a failure pattern $\mathcal{F}$ is said to be correct in $\mathcal{F}$ ($p_i \in \mathit{correct}(\mathcal{F})$). A process is said to be alive until it crashes. Processes that are not correct are called faulty. An environment $\mathcal{E}$ is a set of possible failure patterns. In this paper, we consider every environment, i.e. any number of processes may crash and in particular any process may crash at any time.

A failure detector $\mathcal{D}$ is a distributed oracle that provides the processes with information about failures. A failure detector is defined by its histories. Given a failure pattern $\mathcal{F} \in \mathcal{E}$, a history H of a failure detector $\mathcal{D}$ is a function from $\Pi \times \mathcal{T}$ to $\mathcal{R}_{\mathcal{D}}$, the failure detector range of $\mathcal{D}$, i.e. the set of possible outputs of $\mathcal{D}$: $\mathcal{D}(\mathcal{F})$ denotes a set of failure detector histories that are allowed for $\mathcal{F}$.

An algorithm A is modeled as a set of n deterministic automata, one for every process in the system. A run of A proceeds in steps and at every time t at most one process executes a step. We assume only fair runs, i.e. every correct process

executes infinitely many steps. A step consists of receiving a (possibly empty) message, reading a value of a failure detector, changing the state accordingly, and outputting a (possibly empty) message.

A failure detector is said to solve a problem in a given environment $\mathcal{E}$ if there is an algorithm that solves the problem using message passing and that failure detector (and no other information about failures) for every failure pattern in $\mathcal{E}$. A failure detector $\mathcal{D}$ is said to be stronger than another failure detector $\mathcal{D}'$ in an environment $\mathcal{E}$ if there is an algorithm that uses only $\mathcal{D}$ to emulate the output of $\mathcal{D}'$ for every failure pattern in $\mathcal{E}$. Similarly, detector $\mathcal{D}$ is weaker than $\mathcal{D}'$ in $\mathcal{E}$ if $\mathcal{D}'$ is stronger than $\mathcal{D}$ in $\mathcal{E}$. Failure detector $\mathcal{D}$ is said to be strictly stronger than failure detector $\mathcal{D}'$ in environment $\mathcal{E}$ if $\mathcal{D}$ is stronger than $\mathcal{D}'$ in $\mathcal{E}$ but not vice versa.

The weakest failure detector [6] $\mathcal{D}$ to solve a given problem in an environment $\mathcal{E}$ is a failure detector that is sufficient to solve the problem in $\mathcal{E}$ and that is also necessary to solve the problem, i.e. $\mathcal{D}$ is weaker than any failure detector that solves the problem in $\mathcal{E}$.

We define $\mathcal{D}$ to be (strictly) stronger (resp. weaker) than $\mathcal{D}'$ if $\mathcal{D}$ is (strictly) stronger (resp. weaker) in every environment. Similarly, a weakest failure detector for a problem is defined to be a weakest failure detector for this problem for every environment.

2.2 Set-Agreement

In the set-agreement problem, every process p_i starts with some proposal value v_i and tries to decide a value such that the following three properties are satisfied:

Agreement: At most $n-1$ different values are decided.
Validity: Every value that has been decided must have been a proposal value of some process.
Termination: Eventually, every correct process decides a value.

2.3 Failure Detector $\mathcal{L}$

We now define the *Loneliness* detector $\mathcal{L}$. This failure detector outputs one of two values *"true"* and *"false"*. The intuition behind the semantics of this failure detector is that if the output at some correct process is *"false"* forever, then there is another alive process in the system. By convention, we assume that if a process is crashed at time t, then its failure detector output at time t is *"false"*. The following properties are satisfied:

- at least one process never outputs *"true"*, and
- if only one process is correct, then it eventually outputs *"true"* forever.

More formally:

Definition 1. *The range of $\mathcal{L}$ is $\{$"true", "false"$\}$. For every environment $\mathcal{E}$, for every failure pattern $\mathcal{F} \in \mathcal{E}$, and every history $H \in \mathcal{L}(\mathcal{F})$:*

$$\exists p_i \in \Pi, \forall t, H(p_i, t) \neq \text{"true"} \tag{1}$$

$$\wedge\ \forall p_i \in \Pi, correct(\mathcal{F}) = \{p_i\} \Rightarrow \exists t, \forall t' \geq t, H(p_i, t') = \text{"true"} \tag{2}$$

3 The Sufficient Part

To show that failure detector $\mathcal{L}$ is sufficient to solve set-agreement in our model, we give an algorithm that implements set-agreement with $\mathcal{L}$. The algorithm is depicted in Figure 2.

To ensure that at most $n-1$ proposal values are decided, every process tries to agree with another process on one value. To achieve this, initially some processes send their values. To prevent a circular value exchange, i.e. a situation where the proposal values are simply permuted, the values are only sent to processes with a higher id. This means, that process p_1 sends its value to everybody (except itself), process p_i to all processes from p_{i+1} to p_n, and process p_n to nobody.

If some process receives[1] one of these values, it sends this value to all other processes and decides. As long as there is another correct process, every correct process decides either through one of the messages that were initially sent or, if it does not receive such a message (e.g., because it has a lower id than the other correct processes), it decides through a message of an already decided process. Note that it may be possible that a process receives its initial value back in such a message. In this case, the sender of this message does not decide its own proposal value.

To deal with crashes, we only execute these steps if the output of the failure detector is *"false"*. But in the case of only one correct process in the system, we do not want to wait for messages of other processes forever. Therefore, if the output of the failure detector changes to *"true"* – and by its property (2) in the case of only one correct process it will eventually do so – this process simply decides its own proposal value. We can do this without violating agreement, because by property (1) there will always be one process that does not decide

---

Algorithm for process p_i:

```
1  to propose(v):
2    initially:
3      send ⟨v⟩ to all p_j with j > i;
4    on receive ⟨v'⟩ do:
5      send ⟨v'⟩ to all;
6      decide v'; halt;                          (* decision D1 *)
7    on L = "true" do:
8      send ⟨v⟩ to all;
9      decide v; halt;                           (* decision D2 *)
```

Fig. 2. Implementing set-agreement with $\mathcal{L}$

[1] For simplicity of the presentation, we assume that the code Lines 5-6 and Lines 8-9 are executed atomically.

due to a *"true"* output, and as we have argued before, processes that decide due to a message exchange eliminate at least one value.

Proposition 1. *The algorithm in Figure 2 implements set-agreement in every environment $\mathcal{E}$.*

Proof. We have to prove the three properties of set-agreement, namely agreement, validity, and termination.

Agreement. We start with the agreement property of set-agreement. We assume a run where all processes decide and every process p_i has a distinct initial value v_i. Without this assumption, agreement is trivially met.

By Property (1) of $\mathcal{L}$, not all processes can have decided by decision D2. Therefore, in such a run at least one process decides by D1. This means that it is sufficient to show that if at least one process decides by D1, then at most $n-1$ values are decided.

Among the processes that decide by D1, consider p_i as the process with the highest id and let v' be the decided value. We distinguish between the two cases where p_i decides its initial value ($v' = v_i$), and where it does not.

Case 1: The only possibility that the decided value v' is equal to p_i's value v_i is that a process p_j with $j > i$ has received p_i's initial message and decided v_i. Therefore, p_i and p_j decide the same value and at most $n-1$ values are decided.

Case 2: If v' is not equal to v_i and $i = n$, then v_n will never be decided because process p_n does not send its value to anybody. If $i < n$, then the only possibility that v_i is decided is if a process p_k with $k > i$ has received v_i from p_i and decided by D1. But as p_i is the process with the highest id that decides by D1, such a k does not exist. And therefore, v_i is never decided.

Validity. The validity property of set-agreement is trivially satisfied, since only proposal values are sent.

Termination. If some correct process decides by D1 or D2, then it sends its decided value to all processes and all correct processes that have not yet decided eventually receive this value and also decide.

Therefore, it remains to show that in every run some correct process decides by D1 or D2. We distinguish two cases: the case when there exist at least two correct processes in a run with a failure pattern $\mathcal{F} \in \mathcal{E}$, and the case with only one correct process.

Case 1: If there are at least two correct processes and none decides by D2, then eventually, the one with the highest id receives the initial message of the other ones and decides by D1.

Case 2: If there is only one correct process and it does not decide by D1, then by property (2) of $\mathcal{L}$, this process eventually decides by decision D2. □

4 The Necessary Part

Following the approach of Chandra et al. [6], we show that failure detector $\mathcal{L}$ is necessary to solve set-agreement in our model by providing an algorithm that emulates the output of $\mathcal{L}$ given any algorithm A and failure detector $\mathcal{D}$, such that A using $\mathcal{D}$ solves set-agreement. Figure 3 presents such an emulation algorithm. The output of our emulation of $\mathcal{L}$ is provided through a special variable *output*.

The idea for the emulation of $\mathcal{L}$ is that if all messages that are sent by algorithm A get delayed for a very long time, the safety properties of set-agreement still have to hold, while for the case that only one process is correct, even the liveness property has to hold, i.e. the algorithm has to terminate. Therefore, every process executes A with $\mathcal{D}$, omits to send any messages that are generated by algorithm A to other processes, and outputs *"false"* until A terminates.

Property (1) of $\mathcal{L}$ is thus always fulfilled, because otherwise the executions at all processes would have terminated without ever receiving a message and therefore agreement could not have been guaranteed. But nevertheless, if there is only one correct process p_i, the algorithm A executed at p_i has to terminate and property (2) of $\mathcal{L}$ is also guaranteed.

Interestingly, this technique works for every non-trivial problem in which communication between processes is necessary, i.e. where not all processes may terminate without receiving messages from other processes. Therefore, $\mathcal{L}$ is necessary for all of these problems.

Proposition 2. *The algorithm in Figure 3 implements $\mathcal{L}$ in every environment $\mathcal{E}$.*

Proof. Assume there exists a run r, where the algorithm in Figure 3 does not fulfill property (1) of $\mathcal{L}$ with a failure pattern $\mathcal{F} \in \mathcal{E}$. This means, that in run r, for every process, there exists a time when *output* $=$ *"true"*, i.e. the execution of algorithm A has terminated at all processes without receiving any message from other processes at all.

Let t be the time at which A has terminated at all processes in run r. Then, since the system is totally asynchronous, it is possible to construct a valid run r' of A with the same failure pattern $\mathcal{F}$, where all messages to other processes get delayed to a time after t, and all processes have terminated A at time t.

Algorithm for process p_i:

1 *output* := *"false"*;
2 execute A with value i and detector $\mathcal{D}$, but omit sending messages to others;
3 if A has terminated, then *output* := *"true"*;

Fig. 3. Implementing $\mathcal{L}$ with an algorithm A and a failure detector $\mathcal{D}$ that solve set-agreement

Note that a failure detector is solely specified as a function over a failure pattern in an execution, i.e. it is not allowed to output any information about the state of other processes or to give hints about the proposal values.

Therefore, to fulfill the validity property of set-agreement, the decision value at every process p_i can only be its proposal value i. A contradiction with the agreement property of set-agreement. Therefore, property (1) of $\mathcal{L}$ is always satisfied.

If for some run r of our algorithm, for some process p_i, $\mathcal{F}$ is the failure pattern in run r and $correct(\mathcal{F}) = \{p_i\}$, then it is possible to construct a run r_A of A in which no faulty process is able to send a message (because the system is totally asynchronous) and p_i takes exactly the same steps as in r. By the termination property of set-agreement, eventually algorithm A has to terminate in run r_A at p_i. Since r and r_A are indistinguishable for p_i, it terminates the execution of A also in r and the output changes to *"true"*. Thus, property (2) is also satisfied. □

Theorem 1. *$\mathcal{L}$ is the weakest failure detector for set-agreement in a message passing system.*

Proof. We have shown in Proposition 1 that $\mathcal{L}$ is sufficient and in Proposition 2 that it is necessary for set-agreement in all environments. □

5 Comparing $\mathcal{L}$ and Anti-Ω

To keep our proofs as generic as possible, we first introduce the notion of eventual failure detectors. We say that a failure detector is an eventual failure detector if the detector can behave arbitrarily for any finite amount of time. A more formal definition can be found in [15] where such failure detectors are called strongly unreliable failure detectors.

Zieliński shows in [16] that every eventual failure detector (that satisfies some other assumptions that are irrelevant here) is stronger than anti-Ω, the weakest failure detector for set-agreement in a shared memory [10]. Each query to the anti-Ω detector returns a process id. The failure detector guarantees that there is a correct process whose id will be returned only finitely many times. Clearly, anti-Ω is an eventual failure detector and $\mathcal{L}$ is not. We show that $\mathcal{L}$ is strictly stronger than anti-Ω. This means, that to implement set-agreement in message passing there is a strictly stronger failure detector necessary than in shared memory.

Lemma 1. *$\mathcal{L}$ is stronger than anti-Ω.*

Proof. An implementation of anti-Ω using $\mathcal{L}$ is given in Figure 4. The basic idea is simple: Every process p_i outputs the id j of a process p_j such that j is the lowest id of all processes from which p_i has not yet heard that they have had a *"true"* as failure detector output. For this, the processes remember the ids of processes that have received a *"true"* from $\mathcal{L}$ in a set *lonely*. The output of anti-Ω is emulated in a special variable *output*.

Algorithm for process p_i:

```
1  initially:
2      lonely := ∅;
3      output := {1};
4  on L = "true" do:
5      lonely := lonely ∪ {i};
6      send ⟨lonely⟩ to all;
7      output := min({1, ..., n} \ lonely);
8  on receive ⟨lonely'⟩ do:
9      if lonely ≠ lonely' then send ⟨lonely ∪ lonely'⟩ to all;
10     lonely := lonely ∪ lonely';
11     output := min({1, ..., n} \ lonely);
```

Fig. 4. Implementation of anti-Ω using $\mathcal{L}$

We now show that this transformation indeed emulates anti-Ω. From property 1 of the definition of $\mathcal{L}$, the output of at least one process is never a *"true"*. Therefore, there is always at least one id that is output (i.e. it is never $lonely = \{1, \ldots, n\}$).

To prove that there is a correct process whose id is output only finitely often, note that eventually the set *lonely* is the same at all correct processes because it can only grow and will always be a subset of $\{1, \ldots, n\}$ (and every correct process relays it after every change). Therefore, eventually all correct processes have the same output. Now assume the id of every correct process is output infinitely often at the processes. This implies that there is only one correct process, because all processes always output the minimum of $\{1, \ldots, n\} \setminus lonely$ which can only shrink and therefore never oscillates between different process ids. But from property 2 of the definition of $\mathcal{L}$, a single correct process eventually receives a *"true"* and therefore belongs to its set *lonely*. A contradiction. □

Lemma 2. *No eventual failure detector is stronger than $\mathcal{L}$.*

Proof. Assume there exists an algorithm A that transforms an eventual failure detector $\mathcal{D}$ to $\mathcal{L}$. Then, assume for every $1 \leq i \leq n$, a run r_i of A with failure pattern $\mathcal{F}_i$ and $correct(\mathcal{F}_i) = \{p_i\}$ and where the faulty processes take no steps. If A is correct, then eventually the output at process p_i in run r_i has to be *"true"*, say at time t_i. Similarly, assume a run r of A with a failure pattern $\mathcal{F}$ with $correct(\mathcal{F}) = \Pi$, but no process p_i receives a message from any other process before or at time t_i and every p_i is scheduled as in r_i. Let the output of $\mathcal{D}$ at every process p_i before time t_i be exactly as in run r_i (this is possible, since $\mathcal{D}$ may behave arbitrarily for any finite amount of time). Then, for every process p_i, run r_i is indistinguishable from run r before time t_i and every process p_i outputs *"true"* at time t_i. But this contradicts property 1 of $\mathcal{L}$. □

Theorem 2. *$\mathcal{L}$ is strictly stronger than anti-Ω.*

Proof. Follows directly from Lemma 1 and Lemma 2. □

6 Comparing $\mathcal{L}$ and Σ

We now show that Σ, the weakest failure detector to emulate a shared memory in message-passing systems [12] is strictly stronger than $\mathcal{L}$. In a sense, this indicates that emulating a shared memory in message passing is strictly harder than solving set-agreement, confirming the result of [13]. By convention, we assume that if a process is crashed at time t, then its failure detector output is Π at time t. At each invocation, Σ outputs a list of trusted processes and it satisfies two properties:

Intersection: Given any two lists of trusted processes, possibly at different times and by different processes, at least one process belongs to both lists.
Completeness: Eventually no faulty process is ever trusted by any correct process.

Lemma 3. *Σ is stronger than $\mathcal{L}$.*

Proof. The reduction is simple: At the beginning, every process outputs *"false"*. For every process p_i, if the output of Σ is $\{p_i\}$, output *"true"*.

Assume that for every process there is some time when the output of $\mathcal{L}$ is *"true"*. Since this happens only if at every process p_i, $\{p_i\}$ is output, the intersection property of Σ is clearly violated. Therefore, this will never happen and property 1 of $\mathcal{L}$ is never violated.

From the completeness property follows that if a process p_i is the only correct process, the output will eventually be *"true"* (property 2 of $\mathcal{L}$). □

For the special case that the system consists only of two processes, the specifications of set-agreement and consensus are equivalent. Delporte-Gallet et al. show in [17] that for this case Σ is the weakest failure detector for consensus. Together with Theorem 1 this immediately implies that $\mathcal{L}$ and Σ are also equivalent for this case. However, in the following lemma we show that for $n > 2$ this is not the case.

Lemma 4. *$\mathcal{L}$ is not stronger than Σ, if $n > 2$.*

Proof. Assume there exists an algorithm A that transforms $\mathcal{L}$ into Σ. Let $P = P_1, P_2, P_3$ be any partitioning of Π. Then assume two runs r_1 and r_2 where the processes in P_i are correct in run r_i and all other processes are faulty from the beginning, and the output of $\mathcal{L}$ at the processes in partition P_i is *"true"*. Since A fulfills completeness, it eventually has to output in every run r_i a subset of P_i, say at time t_i.

Now imagine a run r in which the processes in P_1 and P_2 are correct and the output of $\mathcal{L}$ is *"true"*. Additionally, no message of a process from a different partition is received in partition P_1 and P_2 before time t_1 (respectively t_2) and

the messages between the processes in P_1 and P_2 are exactly scheduled as in runs r_1 and r_2. The runs r_1 and r_2 are indistinguishable from run r before time t_1 (respectively t_2). Therefore, the output at time t_i will be a subset of P_i for partition $i = 1, 2$. But this contradicts to the intersection property of Σ. So there exists no such algorithm A. □

Theorem 3. *If $n > 2$, then Σ is strictly stronger than $\mathcal{L}$.*

Proof. Lemma 3 shows that Σ is stronger than $\mathcal{L}$ and Lemma 4 shows that it is strictly stronger. □

7 Summary

We have determined the weakest failure detector for set-agreement in a message-passing system where processes may fail by crashing. The failure detector is called $\mathcal{L}$ and it returns at every invocation *"true"* or *"false"*. It ensures that (1) there is at least one process where the output is always *"false"*, and (2) if there is only one correct process, then the output at this process is eventually *"true"* forever.

Acknowledgments. We are grateful to Sam Toueg for helpful suggestions on the sufficient part of our proof. Furthermore, we would like to thank the reviewers for their helpful comments.

References

1. Chaudhuri, S.: More choices allow more faults: Set consensus problems in totally asynchronous systems. Inf. Comput. 105(1), 132–158 (1993)
2. Saks, M., Zaharoglou, F.: Wait-free k-set agreement is impossible: The topology of public knowledge. SIAM J. Comput. 29(5), 1449–1483 (2000)
3. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for t-resilient asynchronous computations. In: STOC 1993: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, pp. 91–100. ACM, New York (1993)
4. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. Journal of the ACM 46(6), 858–923 (1999)
5. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM 43(2), 225–267 (1996)
6. Chandra, T.D., Hadzilacos, V., Toueg, S.: The weakest failure detector for solving consensus. J. ACM 43(4), 685–722 (1996)
7. Raynal, M., Travers, C.: In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 3–19. Springer, Heidelberg (2006)
8. Guerraoui, R., Herlihy, M., Kouznetsov, P., Lynch, N., Newport, C.: On the weakest failure detector ever. In: PODC 2007: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, pp. 235–243. ACM, New York (2007)

9. Chen, W., Zhang, J., Chen, Y., Liu, X.: Weakening failure detectors for k -set agreement via the partition approach. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 123–138. Springer, Heidelberg (2007)
10. Zieliński, P.: Anti-Omega: the weakest failure detector for set agreement. In: PODC 2008: Proceedings of the twenty-seventh annual ACM symposium on Principles of distributed computing (2008)
11. Zieliński, P.: Anti-Omega: the weakest failure detector for set-agreement. Technical report, UCAM-CL-TR-694 (2007)
12. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R.: Shared memory vs message passing. Technical report, LPD-REPORT-2003-001 (2003)
13. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R.: Sharing is harder than agreeing. In: PODC 2008: Proceedings of the twenty-seventh annual ACM symposium on Principles of distributed computing (2008)
14. Attiya, H., Bar-Noy, A., Dolev, D.: Sharing memory robustly in message-passing systems. J. ACM 42(1), 124–142 (1995)
15. Guerraoui, R.: Indulgent algorithms. In: PODC 2000: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing (2000)
16. Zieliński, P.: Automatic classification of eventual failure detectors. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 465–479. Springer, Heidelberg (2007)
17. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R. (Almost) all objects are universal in message passing systems. In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724. Springer, Heidelberg (2005)