On Properties of the Group Membership Problem

Mikhail Nesterenko* André Schiper†

February 15, 2007

Technical report TR-KSU-CS-2007-01, Kent State University

Abstract. *We study fault tolerance properties of the group membership problem (GMP) in asynchronous systems. This problem requires each process to determine the processes with which it can communicate. We compare the properties of the GMP and consensus. We define* failure sensitivity *as the necessary property of a failure detector to enable a solution to the GMP. We demonstrate that the known consensus failure detectors — Ω, Σ, Σ^ν are fault insensitive and thus insufficient to solve the GMP. In contrast we define a new reachability failure detector $\mathcal{R}$ and show it to be the weakest failure detector to solve the GMP. Furthermore, we demonstrate that $\mathcal{R}$ implements the consensus failure detectors. This shows that the GMP is a strictly stronger problem than consensus with respect to the failure detectors that it requires. We present our findings using the primary partition variant of the GMP. We extend them to the partitionable GMP as well as eventual GMP: a weaker variant of the GMP where a process is allowed to make finitely many mistakes in its group membership output.*

1 Introduction

The group communication problem studies the properties of broadcast message exchanges between a distributed system of processes organized in groups. It is one of the fundamental problems in distributed computing. A large body of literature is devoted to the subject. The reader is referred to the following surveys [6, 7] for a comprehensive literature review. The processes are typically assumed to be asynchronous. The *group membership problem* (GMP) is an integral component of the group communication problem [10, 13] if group membership is dynamic. The GMP deals with presenting a consistent membership set to the processes despite process and link failures. *Primary partition* GMP provides consistent membership information only to the processes in a single partition of the network, the so-called *primary partition* [13]. Some researchers consider this semantics to be too restrictive. They introduce *partitionable* GMP, which provides consistent membership to processes in all partitions of the system [10].

Another well-studied problem of distributed computing is *consensus* [2]. Consensus requires a set of processes to agree on a single common value. Consensus is not deterministically solvable in asynchronous systems even with a single crash-fault [11]. One way to circumvent the impossibility is to relax the problem to *eventual consensus* [15] where each process is allowed to make finitely many errors. Another approach is to introduce *unreliable failure detectors* [5]. Failure detectors

*This author is with Kent State University. This research is supported in part by NSF CAREER Award 0347485 and a grant from the Wright Center for Sensor System Engineering.

†This author is with École Polytechnique Fédérale de Lausanne.

provide processes with information about process failures in the system. This information may not be reliable.

A detector may not be implementable in an asynchronous system. However, a failure detector is a convenient abstraction of the synchrony necessary to solve consensus in the presence of faults. Treated as such, the concept of a detector poses the question of determining the least amount of synchrony needed to solve consensus. Chandra et al introduce the concept of the *weakest failure detector* [3]. An algorithm that solves consensus has to use a detector that is at least as strong as the weakest failure detector. Whether a certain failure detector is the weakest is proven by showing the following properties: (i) there is an algorithm that solves the problem using this detector, and (ii) this detector can be implemented by any detector that solves the problem. The specification of the weakest failure detector may depend on the variant of the consensus problem. Consensus is *uniform* if no two processes can reach a different decision. Consensus is *nonuniform* if only correct processes have to agree on the decision. That is, nonuniform consensus allows the decision of crashed processes to differ from those of correct ones. The *environment* [9] restricts the number of processes that may fail. *Majority* environment means that the majority of the processes in the system are correct while *any* or *arbitrary* environment does not restrict the faults. It is proven [3, 12] that *leader detector* Ω is the weakest failure detector to solve consensus in the majority environment. Delporte et al [8] proved that a combination of Ω with a *quorum failure detector* Σ is the weakest failure detector to solve uniform consensus in any environment. Esler et al [9] showed that Ω with another quorum failure detector Σ^ν is the weakest failure detector for nonuniform consensus in any environment.

It is natural to compare consensus and the GMP as both are agreement problems. Chandra et al [4] addressed this question. They defined a restricted version of the GMP, which they called the WGM, where only two processes are allowed to leave the group and only one process is allowed to fail by crashing. The processes have to agree on a single group membership set. It turns out that, similarly to consensus, the WGM is not solvable in an asynchronous system. Babaoğlu et al [1] considered reachability detector for the GMP.

Recently, Schiper and Toueg [14] further analyzed the relationship between the two problems. They argued that there are two subproblems to the GMP: (i) determining the set of processes that belong to the partition and (ii) ensuring that the partition members agree on this set. Schiper and Toueg show that the second subproblem (agreeing on a *view*) is equivalent to consensus. However, as they demonstrate, this second subproblem is oblivious to the domain of agreement: the agreed set of processes may not correspond to the actual partition membership. As the present study indicates, the first subproblem of the GMP makes the whole problem strictly stronger than consensus.

Our contribution. The results presented in this paper demonstrate that the GMP is strictly stronger than consensus with respect to the power of failure detectors that are required to solve it. Our results apply to both the primary partition and partitionable version of the GMP.

We define the GMP in the most restrictive terms: each run contains at most one fault that disables (crashes) some of the processes. We require a solution to the GMP to have non-trivial safety and liveness properties: each process should never exclude the members of its partition and eventually has to correctly determine them. Our definition of the GMP is stronger than the WGM [4], yet it is restrictive enough to accommodate any practical implementation of the GMP. We also consider a version of the GMP that is weaker than WGM. The *eventual group membership problem* (EGMP) allows each process to make finitely many mistakes with respect to its group membership.

We define the *failure insensitivity* and *strict failure insensitivity* properties of a failure detector. We prove that the GMP and EGMP cannot be solved only with failure insensitive and strictly failure insensitive failure detectors respectively. We show that the known consensus failure detectors: Ω,

Σ, Σ^{ν} are strictly failure insensitive and thus insufficient to solve either the GMP or EGMP. The impossibility applies to primary partition and partitionable variants of GMP and EGMP. Our results undermine the idea of using partitionable semantics as a way to bypass the difficulty of solving agreement problems in the presence of network partitions.

On the constructive side, we define failure sensitive *reachability failure detectors* $\mathcal{R}$, $\Diamond\mathcal{R}$ and show that they are the weakest failure detectors to solve the GMP and EGMP respectively. Moreover, we show that the consensus oracles can be implemented out of the solutions to the primary and partitionable versions of the GMP.

The rest of the paper is organized as follows. We introduce our computation model, the failure detectors and the primary partition version of the GMP in Section 2. We focus on this version of the problem for the major part of the paper. We define the failure sensitivity property of a failure detector and prove it necessary to solve the GMP in Section 3. In Section 4, we present an algorithm $\mathcal{R}2\mathcal{G}$ that solves the GMP using the reachability failure detector. We also show that the reachability failure detector is the weakest detector to solve the GMP. We extend our results to the EGMP and to the partitionable version of the GMP in Sections 5 and 6 respectively. We conclude the paper in Section 7 where we discuss the implications and further directions of our research.

2 Model and Definitions

2.1 System and Faults

A distributed system consists of a set N of processes fully connected by communication channels. Each process has a unique identifier. The distributed system is *asynchronous*: there is no bound on the difference between process speeds. The communication channels are FIFO and reliable. An *algorithm* $\mathcal{A}$ specifies a set of variables and actions for each process of the system. Some of the variables are *output* variables. Each action is guarded by a boolean predicate. Unless otherwise specified, this predicate is defined on the state of the local variables and incoming channels of the process. A *configuration* of $\mathcal{A}$ is a collection of states of all processes and channels of the system. An *action* of the algorithm moves the system from one configuration to another. An action of $\mathcal{A}$ is *applicable* in a certain configuration if its guard evaluates to **true** in this configuration. A *run* ρ of $\mathcal{A}$ is a maximal fair sequence of configurations where each subsequent configuration c_{i+1} is obtained by atomically executing an action applicable in c_i. The maximality of a run means that no run can be a proper prefix of another run. That is, a run either terminates in a configuration where none of the actions are applicable or the run is infinite. The (*weak*) *fairness* means that if an action is applicable in all but finitely many configurations of an infinite run, then this action is executed infinitely often. Observe that a run that contains a fair suffix is also fair. An algorithm *solves* a problem if every run of the algorithm satisfies the properties of the problem specification.

A *crash fault* (or just *fault*) F is a special action that disables all actions of an arbitrary set of processes. The fault action does not belong to a particular process, it is a single action per system. A process whose actions are disabled in a run ρ is *faulty* or *crashed*. A processes p that does not crash in ρ is *correct* in ρ. Unless stated otherwise, P denotes the set of correct processes. Notice that if a process sends a message before it crashes and the recipient is a correct process, then this message is received. For simplicity, we consider a single fault per run. That is, the fault action is applicable in every configuration of the run until it is executed. After the execution of the fault action, it is no longer applicable. Weak fairness of action execution does not apply to the fault action. That is, we also consider runs where no fault occurs.

We use lowercase Latin letters p, p', etc. to denote processes and uppercase letters to denote sets of processes. We use lower case Greek letters to denote configuration sequences and runs. Calligraphic letters as well as uppercase Greek letters denote algorithms and failure detectors. $p \rightsquigarrow p'$ means that there is a path from p to p' in a graph. We use $c.p.v$ to denote the value of the variable v of process p in configuration c. $\sigma|V$ is a projection of sequence σ onto the set of variables V. In particular, $\sigma|output$ is the projection of the sequence on the output variables. Also, if multiple algorithms are run jointly, $\sigma|\mathcal{A}$ is the projection of the sequence onto the variables of algorithm $\mathcal{A}$.

2.2 Group Membership Problem

Note that unless otherwise stated, we discuss the primary partition variant of the GMP. The *group membership problem* (GMP) requires processes to output the set of non-faulty processes. The input to every process is the set of process identifiers N. That is, each process knows all existing identifiers in advance. The output has to conform to the following two properties:

intersection — the output of a process has to contain a (not necessarily strict) superset of correct processes; and

completeness — eventually, each process either crashes or outputs only correct processes.

Traditional definitions of the GMP [6, 7] define successive *views* of the partition membership that processes have to agree to. We simplify the definition such that the processes are requested to output a single view that corresponds to the set of correct processes. Thus, in our definition the view agreement is implicit.

2.3 Failure Detectors

A *failure detector* (or just *detector*) is an algorithm, the guards of whose actions may include failure information. The output variables of a detector supply the failure information for other algorithms to use. If another algorithm $\mathcal{A}$ *uses* a failure detector $\mathcal{FD}$, a run of $\mathcal{A}$ is a weakly fair interleaving of action executions of $\mathcal{A}$ and $\mathcal{FD}$. A failure detector can be treated as an abstract specification. An algorithm $\mathcal{B}$ *implements* a failure detector $\mathcal{FD}$ if the projection of every run of $\mathcal{B}$ onto the variables of $\mathcal{FD}$ is a run of $\mathcal{FD}$.

A failure detector $\mathcal{WFD}$ is *the weakest failure detector* [3] required to solve a problem $\mathcal{P}$ if (i) there is an algorithm $\mathcal{A}$ that solves $\mathcal{P}$ using $\mathcal{WFD}$; and (ii) for every solution $\mathcal{B}$ to $\mathcal{P}$, there exists an algorithm $\mathcal{C}$ that implements $\mathcal{WFD}$ using $\mathcal{B}$. That is, every solution to $\mathcal{P}$ can be used to implement $\mathcal{WFD}$. Suppose $\mathcal{WFD}_1$ and $\mathcal{WFD}_2$ are the weakest failure detectors to solve problems $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively. Problem $\mathcal{P}_1$ is *strictly stronger* than $\mathcal{P}_2$ with respect to the failure detectors that it requires (or just *strictly stronger*) if (a) there is an algorithm that implements $\mathcal{WFD}_2$ using $\mathcal{WFD}_1$; and (b) there is no algorithm that solves $\mathcal{P}_1$ using only $\mathcal{WFD}_2$.

Consensus failure detectors. To each process in the system, the *leader detector* Ω deterministically outputs the identity of a single process in the system. To denote this determinism without restricting the selection procedure we use function *min* that maps a set of processes to one of its elements. The output l of Ω has the following property. For each correct process, the output of Ω converges to the value of *min* for the set of correct processes. Notice that due to the definition of *min*, a correct process eventually becomes the leader of all correct processes.

In *quorum* failure detector Σ, each process outputs a set Q of processes called *quorum*. In Σ, the output quorums possess two properties. According to the *intersection* property, any two quorums of two processes intersect. That is

$$\forall p, p' \in N; \forall c, c' \in \rho : c.p.Q \cap c'.p'.Q \neq \varnothing$$

According to the *completeness* property, eventually the quorum of a correct process contains only correct processes. That is,

$$\forall p \in P, \exists \sigma = \textit{suffix}(\rho) : \forall c \in \sigma, q \in c.p.Q \Rightarrow q \in P$$

A less restrictive version of quorum detector — Σ^{ν} requires that only the quorums of correct processes intersect. Detector Σ^{ν} has the same completeness property as Σ.

Reachability failure detector. The reachability failure detector $\mathcal{R}$ also outputs quorums to each process. Note that the application of term *quorum* to the outputs of $\mathcal{R}$ is somewhat liberal: the quorums of two processes do not have to intersect. This failure detector has the same completeness property as Σ. To state the intersection property of $\mathcal{R}$ we need to introduce additional notation. Fix a set of processes P and a set of quorums $\bar{Q}$ with a single quorum $Q \in \bar{Q}$ for every process $p \in N$. A *reachability graph* is a directed multigraph $R(\bar{Q})$ defined as follows. The nodes of R are processes of P. There is an edge from p to p' in R if the quorum output to p contains p'.

The *intersection property* of $\mathcal{R}$ requires that for an arbitrary set of quorums $\bar{Q}$ output in a run, a correct process p is reachable from any other process p' in $R(\bar{Q})$. Note that p' does not have to be correct. Formally,

$$\forall p \in P, \forall p' \in N; \bar{Q} \equiv \{c''.p''.Q | p'' \in N, c'' \in \rho\} : (p' \rightsquigarrow p) \in R(\bar{Q})$$

3 Failure Sensitivity

Definition 1 A failure detector $\mathcal{I}$ is *failure insensitive* (or just *insensitive*) if there exists a fault F, and two runs ρ_1 and ρ_2 of $\mathcal{I}$ with the following proprieties. Run ρ_1 contains F while ρ_2 does not contain a fault. Runs ρ_1 and ρ_2 share a prefix up to the configuration preceding the fault step. For the set of processes P, projections $\rho_1|P|\textit{output}$ and $\rho_2|P|\textit{output}$ are the same, where *output* are the output variables of $\mathcal{I}$.

Informally, failure insensitivity states that there exists a fault that does not affect the outputs of the failure detector to the correct processes. A detector is *failure sensitive* if it is not failure insensitive. Refer to Figures 3(a) and 3(b) for the illustration of failure insensitivity. Suppose the system contains three processes p_1, p_2 and p_3. Each process uses Ω. Process p_3 is the leader of the system. Thus, eventually Ω outputs p_3 to all processes. Notice that the output of correct processes does not change in case one of the non-leader processes is faulty. Consider the outputs of Σ in the same system. Let process quorums be $\{p_1\}$, $\{p_1, p_2\}$, and $\{p_1, p_3\}$ respectively. In compliance with the intersection property of Σ, all quorums intersect: they contain p_1. If either p_2 or p_3 are faulty, the quorums still satisfy the specification of Σ.

In contrast, consider the system that uses a failure sensitive detector $\mathcal{R}$ instead. Refer to Figures 3(c) and 3(d) for illustration. If all processes are correct, then, according to the intersection property, in the reachability graph formed by the output quorums, the processes should be mutually reachable. However, if p_2 crashes, then, due to completeness property, the quorums of the remaining processes should eventually exclude p_2.

Lemma 1 *The leader failure detector Ω is failure insensitive.*

Proof: Refer to Figure 3(a) for an illustration to the proof. Let p be the leader of the whole system. Observe that there exists at least one fault F that creates the set of correct processes P such that the leader of the correct processes is also p. That is $min(P) = min(N) = p$.

$\mathcal{I}$	separation insensitive failure detector
$\mathcal{A}$	assumed algorithm that solves the GMP using only $\mathcal{I}$
F, P	resp. a fault and the set of correct processes it creates
ρ_1, ρ_2	runs of $\mathcal{I}$ with the same output in P; ρ_1 has F, ρ_2 has no faults
σ_1	run of $\mathcal{A}$ whose projection is ρ_1
σ_2	constructed run of $\mathcal{A}$ whose projection is ρ_2
c_f	configuration of σ_1 immediately preceding the execution of F
c_o	configuration of σ_1 where a process of P outputs P.
σ'	projection of segment of σ_1 from c_f to c_o onto P
σ''	σ' with actions of ρ_1 replaced by those of ρ_2
σ'''	expanded σ'' to include variables of $\mathcal{I}$ outside of P
σ''''	expanded σ''' to include variables of $\mathcal{A}$ outside of P

Figure 1: Notation for the proof of Theorem 1.

Let us consider a run ρ_2 of Ω that does not contain a fault. According to the property of Ω, ρ_2 contains a suffix where in every configuration for each process $p' \in N$, $l.p' = p$. Let us consider the prefix of ρ_2 that ends in such a configuration. We construct another run ρ_1 as follows. The new run contains this prefix of ρ_2, the fault step and no more steps. Observe that after the fault, for every correct process $p'' \in P$, $l.p'' = p$. Hence, the constructed run conforms to the specification of Ω. Notice, however, that the outputs of every process of P in ρ_1 and ρ_2 are identical. Thus, Ω is failure insensitive. $\Box$

The following proposition states the relation of failure sensitivity to other failure detectors. Its proof is similar to that of Lemma 1.

Proposition 1 *Quorum failure detectors Σ and Σ^ν are failure insensitive while reachability failure detector $\mathcal{R}$ is failure sensitive.*

Theorem 1 *A solution to the group membership problem requires a failure sensitive failure detector.*

Proof: The notation for the proof is summarized in Figure 1. Assume the opposite: there is an algorithm $\mathcal{A}$ that solves the GMP using only a failure insensitive detector $\mathcal{I}$. By the definition of failure insensitivity, there exist two runs ρ_1 and ρ_2 of $\mathcal{I}$ and a fault F that creates a set of correct processes P in ρ_1 such that ρ_1 and ρ_2 share a prefix up to the configuration where F occurs and where the outputs of $\mathcal{I}$ in P are the same in ρ_1 and ρ_2.

Let us consider an arbitrary run σ_1 of $\mathcal{A}$ whose projection on $\mathcal{I}$ is ρ_1. Recall that σ_1 as well as ρ_1 contain fault F. Let c_f be the last configuration of σ_1 before F is executed. Let us focus on the correct processes after the fault F. That is the processes in set P. Since $\mathcal{A}$ is a solution to the GMP, eventually, every process of P outputs P. Let c_o be the configuration of σ_1 where one of correct processes $p \in P$ outputs P.

We construct another sequence σ_2 of configurations of $\mathcal{A}$ that violates the intersection property of the GMP and show that σ_2 is a valid run of $\mathcal{A}$. Recall that for a configuration sequence to be a run, each subsequent configuration has to be produced by executing an applicable action and the action executions have to be weakly fair. Let σ_2 have no fault. Let it also have the same prefix as σ_1 up to the configuration c_f. We now describe the construction of the next segment of σ_2. The intent is to build a run without a fault, yet where the behavior of the failure detector makes $\mathcal{A}$ in a process of P produce that same output as in σ_1.

Project the segment of σ_1 from c_f to c_o onto P. Denote this projection σ'. Since $\mathcal{I}$ is separation insensitive, the projection $\rho_1|P|output$ and $\rho_2|P|output$ are the same. This means that if there is an action execution in $\rho_1|P$ that changes the output variables, there is an action execution in $\rho_2|P$ that causes a similar change. Let us call the action executions in such a pair *matching*. Note that the matching actions do not have to be identical so long as the change in output they produce is the same. Construct σ'' by replacing each such action of ρ_1 in σ' with a corresponding matching action. Observe that after this replacement, the values of the output variables of $\mathcal{I}$ remain the same. Hence, $\sigma''|\mathcal{A} = \sigma'|\mathcal{A}$. That is, each action of $\mathcal{A}$ in σ'' is executed in an applicable configuration. Recall that one of the processes of $\mathcal{A}$ in the last configuration of σ'' outputs P.

Let ρ'_2 be the segment of ρ_2 that contains the actions of σ''. That is, $\rho'_2|P = \sigma''|\mathcal{I}$. Construct a sequence σ''' by adding the variables of $\mathcal{I}$ and actions of ρ'_2 to σ'' in such a manner that the failure detector actions of σ''' project to ρ'_2. That is, $\sigma'''|\mathcal{I} = \rho'_2$. Thus, in σ''', each action of $\mathcal{I}$ is executed in a configuration where this action is applicable. Notice that actions of $\mathcal{I}$ of processes outside of P and $\mathcal{A}$ in P operate on disjoint set of variables. Hence, σ''' retains the property of σ'' that each action of $\mathcal{A}$ is executed in a configuration where it is applicable. Let σ'''' be σ''' where the remaining variables of $\mathcal{A}$ are appended to each configuration. Specifically, the states of the variables outside of P. The states of these variables are the same as in c_f. Append σ'''' to the constructed prefix of the σ_2. Complete σ_2 by executing the applicable actions of $\mathcal{I}$ and $\mathcal{A}$ in an arbitrary fair manner.

Let us consider the constructed sequence σ_2. Every action in this sequence is executed in an applicable configuration. This sequence contains a fair suffix. Hence, it is fair. Then, by definition, σ_2 is a valid run of $\mathcal{A}$ using $\mathcal{I}$. This sequence does not have a fault. However, σ_2 contains a configuration where a process outputs $P \subset N$. Therefore, this run violates the intersection property of the GMP which means, contrary to our initial assumption, that $\mathcal{A}$ cannot be a solution to this problem. The theorem follows. $\square$

The below corollary follows from Lemma 1, Proposition 1 and Theorem 1.

Corollary 1 *There exists no algorithm that solves the group membership problem using only leader Ω and quorum Σ, Σ^ν failure detectors.*

4 Using Reachability Failure Detector to Solve Group Membership

We start this section by describing an algorithm $\mathcal{R}2\mathcal{G}$ that solves the GMP using $\mathcal{R}$. This algorithm makes use of the concept of a knot in a graph. A *knot* in a graph is its connected component with no outgoing edges. That is, a set of nodes K in a graph constitute a knot if every two nodes of K are mutually reachable in the graph and no node outside of K is reachable from a node of K. Observe that if a set of nodes in a graph has no outgoing edges and no sinks, then this set contains a knot.

The pseudocode of algorithm $\mathcal{R}2\mathcal{G}$ is shown in Figure 2. The implementation of $\mathcal{R}2\mathcal{G}$ is as follows. Each process p has access to the quorum QR output by $\mathcal{R}$. Each process outputs the quorum QG which is the output of the GMP. In array Q, every process maintains the most recent quorum it receives from each process. Initially, every element in the array contains N. There are two actions in $\mathcal{R}2\mathcal{G}$. If $\mathcal{R}$ outputs a new quorum to p, then p broadcasts it to all the other processes. If p receives a new quorum, it is stored in Q. Function *update_QG* is invoked by both actions. This function checks if there is a knot in Q and updates QG if a knot is detected.

Theorem 2 *Algorithm $\mathcal{R}2\mathcal{G}$ solves the group membership problem (GMP) using the reachability failure detector $\mathcal{R}$.*

```
process p
variables
        $QR$ stores output of $\mathcal{R}$ at $p$
        $QG$ outputs of the GMP, initially $\perp$
        $Q[N]$, array of received quorums for each process in $N$
                        initially all $N$
actions
        $Q[p] \neq QR \longrightarrow$
                $Q[p] := QR$
                send $QR$ to all processes of $N$
                $update\_QG$

        receive $QRq$ from $q \longrightarrow$
                $Q[q] := QRq$
                $update\_QG$

function $update\_QG$
        if there is a knot $K$ in the reachability graph $R(Q)$ such that $p \in K$ then
                $QG := K$
```

Figure 2: Algorithm $\mathcal{R}2\mathcal{G}$ solving the GMP using $\mathcal{R}$.

Proof: To prove the theorem we have to show that the runs produced by $\mathcal{R}2\mathcal{G}$ satisfy the two properties of the GMP. Let us analyze the contents of Q. Notice that by the construction of the algorithm, Q contains quorums that were output to the processes of the system by $\mathcal{R}$ in configurations earlier in the run. Notice also that since the elements of Q are never empty the reachability graph $R(Q)$ does not contain sinks.

Observe that according to the intersection property of $\mathcal{R}$, for any process p, if a process p' is correct then p' is reachable from p in $R(Q)$. This means that any knot in $R(Q)$ that contains p also contains every correct process. Hence, if p outputs QG, it is a superset of correct processes. Therefore, $\mathcal{R}2\mathcal{G}$ satisfies the intersection propriety of the GMP.

Let us address the completeness property of the GMP. Due to the completeness property of $\mathcal{R}$, for every process $p' \in P$, its $p'.QR$ eventually contains only correct processes. When $\mathcal{R}$ outputs this quorum to p', process p' sends a message to all processes of N. Correct processes receive messages from other correct processes. Thus, every correct process p eventually gets this message.

Consider a configuration after p receives messages from all members of P where every such message carries the quorum that is a subset of P. Process p stores these quorums in Q. Note, that this means that there are no outgoing edges from processes of P in $R(Q)$. Recall that $R(Q)$ does not contain sinks. Thus, P contains a knot which is output in QG. This satisfies the completeness property of the GMP.

Since, any run produced by $\mathcal{R}2\mathcal{G}$ satisfies both properties of the GMP, $\mathcal{R}2\mathcal{G}$ solves the GMP. $\square$

The weakest failure detector for group membership. To facilitate the construction of the weakest failure detector, we restate the GMP as a failure detector specification $\mathcal{GMP}$. Indeed, $\mathcal{GMP}$ outputs quorums to each process in the system. It has the same completeness property as Σ. The intersection property of $\mathcal{GMP}$ is stated as follows. The quorum of every process contains all correct processes. That is

$$\forall p \in N, \forall p' \in P, \forall c \in \rho : p' \in c.p.Q$$

Notice that for a pair of processes the quorums of Σ may intersect on an arbitrary process. The quorums of $\mathcal{GMP}$ are more restrictive: the quorum of two processes have to intersect on all correct processes. Observe that $\mathcal{GMP}$ is a restriction of $\mathcal{R}$. That is, every run of $\mathcal{GMP}$ is also a run of $\mathcal{R}$. In other words, $\mathcal{GMP}$ implements $\mathcal{R}$. Thus, any algorithm that solves the GMP, implements $\mathcal{R}$. Hence the following theorem.

Theorem 3 *The reachability failure detector $\mathcal{R}$ is the weakest failure detector to solve the group membership problem.*

The GMP and consensus. Observe that a solution to the GMP can be used to implement the consensus failure detectors. Indeed, $\mathcal{GMP}$ has the same completeness property as Σ and the intersection property of $\mathcal{GMP}$ is more restrictive than that of Σ. Thus, $\mathcal{GMP}$ implements Σ. Since Σ^ν is more general than Σ, $\mathcal{GMP}$ also implements Σ^ν. To implement Ω using $\mathcal{GMP}$, it is sufficient to apply the leader function *min* to the quorums output to each process by the $\mathcal{GMP}$. Since for each correct process, the quorum of $\mathcal{GMP}$ converges to the correct process set P, the output of *min* for each correct process eventually becomes a single process in P. The combination of the above discussion with Corollary 1 and Theorem 3 yields the following corollary.

Corollary 2 *The group membership problem is strictly stronger than consensus with respect to the failure detectors that it requires.*

5 Extension to Eventual Group Membership

The *eventual group membership problem* (EGMP) requires the satisfaction of the completeness property of the GMP but transforms the intersection property into the second liveness one by requiring that it hold only eventually. That is, each run of a solution to the EGMP should contain a suffix where in every configuration each process either has either crashed or it outputs a superset of correct processes. Unlike the GMP, the output of processes in a run of the EGMP may exclude some correct processes. However, such exclusion may appear only in finitely many configurations of the run. Notice that a solution to the GMP is trivially a solution to the EGMP.

In Section 3, we showed that a failure insensitive detector does not enable a solution to the GMP. We also showed that the known consensus failure detectors are failure insensitive. Yet, the EGMP is a less restrictive problem. This poses a question whether there exists an algorithm that solves the EGMP using only failure insensitive detectors. Recall that a failure insensitive detector $\mathcal{I}$ has two runs ρ_1 and ρ_2. The run ρ_1 has a fault while ρ_2 does not. Runs ρ_1 and ρ_2 share a prefix up to the fault step. The output to correct processes in ρ_1 is identical in ρ_1 and ρ_2. However, an EGMP solution is allowed to make mistakes. The output of $\mathcal{I}$ in ρ_1 and ρ_2 may differ for a process $p' \notin P$. Thus, this process p' may be able to differentiate between ρ_1 and ρ_2 and inform the other processes of the system. An algorithm $\mathcal{A}$, may require each process p of P to output P unless it receives a message from p'. In the latter case p should output N. In other words, if the fault happens, and p does not receive a message from p', then p's output is P. If the fault does not happen, then p eventually gets a message from p' and changes its output to N. This satisfies the requirements of the EGMP. However, there is a stricter definition of insensitivity that still encompasses the consensus failure detectors and prohibits the existence of a solution to the EGMP.

Definition 2 A failure detector $\mathcal{SI}$ is *strictly failure insensitive* if there exists a run ρ_2 of $\mathcal{SI}$ with no faults and with the following property. For every prefix π of ρ_2 there exists a run ρ_1 with a fault F such that, ρ_1 has the same prefix π as ρ_2 and for the set P of correct processes of ρ_2, projections $\rho_1|P|output$ and $\rho_2|P|output$ are the same, where *output* are the output variables of $\mathcal{SI}$.

Intuitively, strict failure insensitivity precludes processes from discovering that the run does not contain a fault: after any prefix π the run may or may not contain a fault. Observe that if a failure detector is strictly failure insensitive, it is also (simply) failure insensitive.

Lemma 2 *Failure detectors Ω, Σ and Σ^ν are strictly failure insensitive.*

The proof of Lemma 2 is given in the Appendix. It is similar to the proof of Lemma 1.

Theorem 4 *A solution to the the eventual group membership problem (EGMP) requires a failure detector that is not strictly failure insensitive.*

The proof of Theorem 4 is given in the Appendix. It is similar to the proof of Theorem 1. The below corollary follows from Lemma 2 and Theorem 4.

Corollary 3 *There exists no algorithm that solves the eventual group membership problem using only the leader Ω and quorum Σ, Σ^ν failure detectors.*

An *eventual failure detector* $\Diamond\mathcal{R}$ has the same completeness property as $\mathcal{R}$ but requires that the intersection property of $\mathcal{R}$ hold only in a suffix of a run. Notice that algorithm $\mathcal{R}2\mathcal{G}$ presented in Section 4 solves the EGMP if it uses $\Diamond\mathcal{R}$ instead of $\mathcal{R}$. Hence the following proposition.

Proposition 2 *There exists an algorithm that solves the eventual group membership problem EGMP using the eventual reachability failure detector $\Diamond\mathcal{R}$.*

Observe that, similar to GMP, EGMP may be restated as an oracle specification $\Diamond\mathcal{GMP}$. Notice that this specification is a restriction of $\Diamond R$. Thus, every algorithm that solves the EGMP, also implements $\Diamond R$. Hence the proposition.

Proposition 3 *The eventual reachability failure detector $\Diamond\mathcal{R}$ is the weakest failure detector to solve the eventual group membership problem.*

6 Extension to Separation Fault and Partitionable Group Membership

6.1 Partitionable Fault Model

A *separation fault* F is a special action that arbitrarily divides the set of processes into several mutually separate *partitions* $\{P_1, P_2, \cdots\}$. Unlike crash fault, a separation fault does not disable process actions. Rather, it prevents communication between separated processes. After the fault, process p successfully sends a message to process p' only if p and p' are in the same partition. Messages sent to separated processes are lost. Notice that a message is still received if it is already in the channel between the processes when the fault separates them. Again, we consider a single separation fault per run.

Since we introduce a different variant of the GMP, in this section we use ptGMP and ppGMP to disambiguate partitionable and primary partition variants of the GMP. The output of the solution to *partitionable GMP* (ptGMP) has to conform to the following properties: *intersection* — the output of each process is a (not necessarily strict) superset of its partition; and *completeness* — eventually, the output of each process contains only the members of its partition.

To make our observations for partitionable fault model non-trivial we have to extend the definitions of failure detectors. Consider a leader failure detector. One implementation may output the partition leader to every processes in one partition P_i and output nothing to the processes in any other partition P_j. This means that the processes of P_j have to discover their partition membership without any help from the failure detector. The impossibility of this task easily follows from Chandra et al [4]. However, we would like to study deeper relationships between the GMP and failure detectors. For that we extend the definition of a failure detector in an intuitive manner. To distinguish the extended definition from classic failure detectors, we call the new constructs *oracles*.

To each process in the system, *leader oracle* Ω^o outputs the identity of a single process in its partition. For each process, the output of the leader oracle eventually converges to the value of *min* for this partition. *Quorum* oracle Σ^o outputs quorums to each process. In Σ^o, the output quorums possess two properties. According to the *intersection* property, any two quorums of two processes of the same partition intersect. According to the *completeness* property, eventually the quorum of a process contains only the processes of its own partition. The reachability oracle $\mathcal{R}^o$ also outputs quorums to each process. This oracle has the same completeness property as Σ^o. The *intersection property* of $\mathcal{R}^o$ requires that in an arbitrary set of quorums $\bar{Q}$ output in a run, two processes p and p' of the same partition are mutually reachable in the reachability graph $R(\bar{Q})$.

6.2 Separation Insensitivity and Partitionable Group Membership

Definition 3 An oracle $\mathcal{I}$ is *separation insensitive* if there exists a separation fault F, an integer i and two runs ρ_1 and ρ_2 of $\mathcal{I}$ with the following proprieties. Run ρ_1 contains F while ρ_2 does not contain a fault. Runs ρ_1 and ρ_2 share a prefix up to the configuration preceding the fault step. In partitioning $\{P_1, P_2, \cdots, P_i, \cdots\}$ resulting from F, projections $\rho_1|P_i|output$ and $\rho_2|P_i|output$ are the same, where *output* are the output variables of $\mathcal{I}$.

Refer to Figure 4 for the illustration of oracle separation insensitivity (cf. Figure 3). Consider the quorums of Σ^o in Figure 4(a). There are no faults and all quorums output to the processes p_1, p_2 and p_3 intersect on process p_1. In case p_2 is separated from the rest of processes (see Figure 4(b)), the processes p_1 and p_3 of one of the resultant partitions do not have to change their quorums. In contrast, consider the quorums of $\mathcal{R}^o$ in Figure 4(c). In case of a similar separation fault, the processes in both partitions have to change their quorums (see Figure 4(d)).

The proof for the following proposition and theorem are given in the Appendix. The proofs are similar to those of the primary partition analogues.

Proposition 4 *The leader oracle Ω^o and quorum oracle Σ^o are separation insensitive while the reachability oracle $\mathcal{R}^o$ is separation sensitive.*

Theorem 5 *A solution to the partitionable group membership problem (ptGMP) requires a separation sensitive oracle.*

Corollary 4 *There exists no algorithm that solves the partitionable group membership problem ptGMP using only leader and quorum oracles.*

Observe that the algorithm $\mathcal{R}2\mathcal{G}$ can solve the ptGMP using $\mathcal{R}^o$ with minor modifications. Hence the following proposition

Proposition 5 *There exists a solution to the partitionable group membership problem ptGMP that uses the reachability oracle $\mathcal{R}^o$.*

Observe that, similarly to the ppGMP, the ptGMP can be relaxed to eventual partitionable group membership. Naturally, results analogous to those of ppEGMP can be obtained for this problem as well.

7 Further Extensions

We would like to conclude the paper with a couple of research directions opened by this paper. Observe that primary partition GMP assumes crashed processes cease to operate. Thus, it cannot be directly applied to separation faults. However, we would like to mention one extension of ppGMP that handles such faults. In the *algorithm's choice* primary partition GMP (ppGMPa) the solution to the GMP chooses a single *live* partition where the completeness property is to be satisfied. The intersection property should be satisfied in every partition.

Investigating the relationship between ppGMPa and consensus is not straightforward: there does not seem to be a single identifiable property, such as separation sensitivity, that differentiates the two problems. However, we believe that the main negative of result of this paper still applies. Thus, we would like to state the following conjecture [1].

Conjecture 1 *The algorithm's choice primary partition group membership problem ppGMPa is strictly stronger than consensus with respect to the oracles it requires.*

In this paper we demonstrated the limit of applicability of well-studied consensus failure detectors to solve the GMP. On the one hand, this stresses the point that a failure detector tends to be a tool that is specialized to solve a particular problem for which it is designed. However, it will be interesting to find out the specific bounds of applicability of the failure detectors, such as Σ and Ω, to a GMP-like problem. That is, our research opens the question, as to what is the strongest problem in group membership that the weakest consensus failure detectors can still solve.

Acknowledgment. We would like to thank Thomas Clouser for his comments and help in improving this paper.

References

[1] O. Babaoğlu, R. Davoli, and A. Montresor. Group membership and view synchrony in partitionable asynchronous distributed systems: specifications. *SIGOPS Operating Systems Review*, 31(2):11–22, 1997.

[2] M. Barborak, M. Malek, and A. Dahbura. The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220, 1993.

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

[4] T.D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *Proceedings of PODC*, pages 322–330, New York, USA, May 1996. ACM.

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

[6] G.V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 4(33):1–43, 2001.

[7] X. Défago, A. Schiper, and P. Urban. Totally Ordered Broadcast and Multicast Algorithms: Taxonomy and Survey. *ACM Computing Surveys*, 4(36):1–50, December 2004.

[1] An argument for this conjecture is given in the Appendix.

[8] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. Shared memory vs message passing. Technical Report LPD-REPORT-2003-001, EPFL, December 2003.

[9] J. Eisler, V. Hadzilacos, and S. Toueg. The weakest failure detector to solve nonuniform consensus. In *Proceedings of PODC*, pages 189–196, Las Vegas, NV, July 2005. ACM.

[10] A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partitionable group communication service. In *Proceedings of PODC*, pages 53–62, August 1997.

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

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

[13] A. Schiper. Dynamic group communication. *Distributed Computing*, 18(5):359–374, 2006.

[14] A. Schiper and S. Toueg. From Set Membership to Group Membership: A Separation of Concerns. *IEEE Transactions on Dependable and Secure Computing*, 3(2):2–12, 2006.

[15] S. Toueg. Personal communication. 1996.

Figures

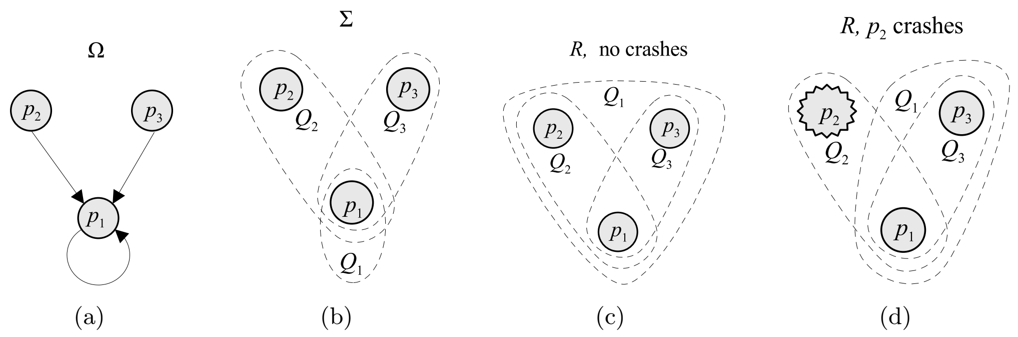

Figure 3: Failure detectors' relation to failure insensitivity. Subfigure (a) shows outputs of Ω. Arrows indicate leader output at each process. Subfigures (b), (c) and (d) show quorum oracles. Dashed lines indicate quorum outputs to each process.

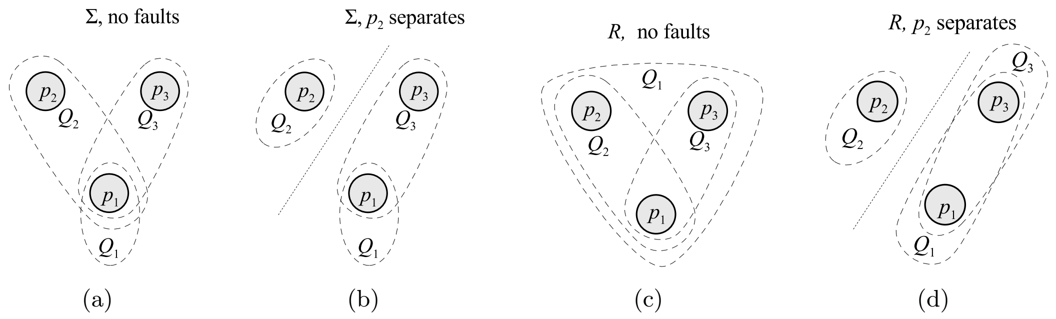

Figure 4: Oracle relation to separation insensitivity. Subfigures (a) and (b) illustration separation insensitivity of Σ^o and contrast it with separation sensitivity of $\mathcal{R}^o$ in subfigures (c) and (d). The dotted line indicates a separation fault.

Appendix

Proofs of Crash Fault Theorems and Lemmas

Proof: (of Lemma 2) We describe the proof of the lemma only for Ω. The proof for Σ and Σ^{ν} is similar. Let ρ_2 be a fault-less run of Ω. Refer to Figures 3(a) for illustration. Let π be an arbitrary prefix of ρ_2. According to the specification of Ω, run ρ_2 contains a suffix where in every configuration all processes of the system output the identity of the leader. Let π' be the prefix that contains the first such configuration. Consider a run ρ_1 of Ω that consists of the longest of π and π' and a fault step F. This fault step is such such that the leader of the system does not fail. That is, for the set of correct processes P in ρ_1, $min(N) = min(P)$. Observe that ρ_1 satisfies the specification of Ω. Thus, runs ρ_1 and ρ_2 satisfy the conditions of Definition 2. This proves that Ω is strictly separation insensitive. □

Proof: (of Theorem 4) Assume, for the sake of contradiction, that there is an algorithm $\mathcal{A}$ that solves the EGMP using only a strictly separation insensitive failure detector $\mathcal{SI}$. Let run ρ_2 be as described in Definition 2. Let us consider all runs of $\mathcal{A}$ whose projections on $\mathcal{SI}$ is ρ_2. Recall that ρ_2 does not have a fault. Since $\mathcal{A}$ is a solution to the EGMP, every such run of $\mathcal{A}$ has a suffix where, in every configuration, the output of each process in the system is N. That is, every such run contains only a finite erroneous prefix where a process may output a value other than N. Let us select a run σ_2 with the longest erroneous prefix. Let c_n be the first configuration in the suffix of σ_2 where every process continuously outputs N. Recall that the projection of σ_2 on $\mathcal{SI}$ is ρ_2. Let d_n be a configuration of ρ_2 that is a projection of c_n.

By the definition of strict separation insensitivity, there is run ρ_1 of $\mathcal{SI}$ with a fault F that shares with ρ_1 a prefix up to d_n. Moreover, ρ_1 has the same outputs as ρ_2 in the set correct processes P created by the fault F. Let us consider a run σ_1 whose projection is ρ_1. Since $\mathcal{A}$ is the solution to the EGMP, σ_1 has a suffix where every correct process outputs $P \subset N$. Since each process outputs N in c_n, this suffix starts further in the run.

The remainder of the proof proceeds similar to the proof of Theorem 1. By rearranging action executions in σ_1, we construct a run σ_2' whose projection is ρ_2. That is, σ_2' does not have a fault. This new run shares a prefix with σ_2 up to configuration c_n. However, it has a configuration that follows c_n where a process of P outputs $P \subset N$. That is, σ_2' has a prefix with erroneous output that is longer than such prefix of σ_2. Yet, σ_2 is selected such that it has the longest such prefix among the runs whose projection is ρ_2. We thus arrive at a contradiction which proves the theorem. □

Proofs of Separation Fault Theorems and Lemmas

Proof: (of Theorem 5) The notation for the proof is summarized in Figure 5. Assume the opposite: there is an algorithm $\mathcal{A}$ that solves the ptGMP using only a separation insensitive oracle $\mathcal{I}$. By definition of separation insensitivity, there exist two runs ρ_1 and ρ_2 of $\mathcal{I}$ and a fault F that creates a partition P such that ρ_1 and ρ_2 share a prefix up to the configuration where F occurs and where the outputs of $\mathcal{I}$ in P are the same in ρ_1 and ρ_2.

Let us consider an arbitrary run σ_1 of $\mathcal{A}$ whose projection on $\mathcal{I}$ is ρ_1. Recall that σ_1 as well as ρ_1 contain fault F. Let c_f be the last configuration of σ_1 before F is executed. Let us focus on partition P created by F. Since $\mathcal{A}$ is a solution to the ptGMP, eventually, every process of P outputs P. Let c_o be the configuration of σ_1 where one of the processes $p \in P$ outputs P.

We construct another sequence σ_2 of configurations of $\mathcal{A}$ that violates the intersection property of the ptGMP and show that σ_2 is a valid run of $\mathcal{A}$. Recall that for a configuration sequence to be a run, each subsequent configuration has to be produced by executing an applicable action and the

$\mathcal{I}$	separation insensitive oracle
$\mathcal{A}$	assumed algorithm that solves the ptGMP using only $\mathcal{I}$
F, P	resp. a fault and one of the partitions created by it
ρ_1, ρ_2	runs of $\mathcal{I}$ with the same output in P; ρ_1 has F, ρ_2 has no faults
σ_1	run of $\mathcal{A}$ whose projection is ρ_1
σ_2	constructed run of $\mathcal{A}$ whose projection is ρ_2
c_f	configuration of σ_1 immediately preceding the execution of F
c_o	configuration of σ_1 where a process of P outputs P.
σ'	projection of segment of σ_1 from c_f to c_o onto P
σ''	σ' with actions of ρ_1 replaced by those of ρ_2
σ'''	expanded σ'' to include variables of $\mathcal{I}$ outside of P
σ''''	expanded σ''' to include variables of $\mathcal{A}$ outside of P

Figure 5: Notation for the proof of Theorem 5.

action executions have to be weakly fair. Let σ_2 have no fault. Let it also have the same prefix as σ_1 up to the configuration c_f. We now describe the construction of the next segment of σ_2. The intent is to build a run without a fault, yet where the behavior of the oracle makes $\mathcal{A}$ in a process of P produce that same output as in σ_1.

Project the segment of σ_1 from c_f to c_o onto P. Denote this projection σ'. Since $\mathcal{I}$ is separation insensitive, the projection $\rho_1|P|output$ and $\rho_2|P|output$ are the same. This means that if there is an action execution in $\rho_1|P$ that changes the output variables, there is an action execution in $\rho_2|P$ that causes a similar change. Let us call the action executions in such a pair *matching*. Note that the matching actions do not have to be identical as long as the change in output they produce is the same. Construct σ'' by replacing each such action of ρ_1 in σ' with a corresponding matching action. Observe that the values of output variables of $\mathcal{I}$ remain the same after replacement. Hence, $\sigma''|\mathcal{A} = \sigma'|\mathcal{A}$. That is, each action of $\mathcal{A}$ in σ'' is executed in an applicable configuration. Recall that one of the processes of $\mathcal{A}$ in the last configuration of σ'' outputs P.

Let ρ'_2 be the segment of ρ_2 that contains the actions of σ''. That is, $\rho'_2|P = \sigma''|\mathcal{I}$. Construct a sequence σ''' by adding the variables of $\mathcal{I}$ and actions of ρ'_2 to σ'' in such a manner that the oracle actions of σ''' project to ρ'_2. That is, $\sigma'''|\mathcal{I} = \rho'_2$. Thus, in σ''', each action of $\mathcal{I}$ is executed in a configuration where this action is applicable. Notice that actions of $\mathcal{I}$ of processes outside of P and $\mathcal{A}$ in P operate on disjoint set of variables. Hence, σ''' retains the property of σ'' that each action of $\mathcal{A}$ is executed in a configuration where it is applicable. Let σ'''' be σ''' where the remaining variables of $\mathcal{A}$ are appended to each configuration. Specifically, the states of the variables outside of P. The states of these variables are the same as in c_f. Append σ'''' to the constructed prefix of the σ_2. Complete σ_2 by executing the applicable actions of $\mathcal{I}$ and $\mathcal{A}$ in an arbitrary fair manner.

Let us consider the constructed sequence σ_2. Every action in this sequence is executed in an applicable configuration. This sequence contains a fair suffix. Hence, it is fair. Then, by definition, σ_2 is a valid run of $\mathcal{A}$ using $\mathcal{I}$. This sequence does not have a fault. However, σ_2 contains a configuration where a process outputs $P \subset N$. Therefore, this run violates the intersection property of the ptGMP which means, contrary to our initial assumption, that $\mathcal{A}$ cannot be a solution to this problem. The theorem follows. $\square$

Separation Fault and Primary Partition Group Membership

Recall that ppGMP assumes crashed processes cease to operate. Thus, ppGMP cannot be directly applied to the separation fault. However, we would like to discuss one extension of ppGMP to handle separation faults. In the *algorithm's choice* primary partition GMP (ppGMPa) the solution to the GMP chooses a single *live* partition where the completeness property is to be satisfied. The intersection property should be satisfied in every partition.

Note that there has to be a deterministic mapping between a fault and the algorithm's live partition selection. If the mapping is not deterministic (i.e. the algorithm may select different live partitions in different runs with the same fault), then the algorithm may have a run where there are multiple live partitions or no live partition at all.

There does not seem to be a single identifiable property that separates the ppGMPa and consensus. Due to space limitations we could not provide a rigorous proof for the following statement. Thus, we are presenting it as a conjecture.

Conjecture 2 *The algorithm's choice primary partition group membership problem ppGMPa is strictly stronger than consensus with respect to the oracles it requires.*

We would like to give an intuition as to why we believe it to be true. Neither Σ^o nor Ω^o appear sufficient to enable a solution to the ppGMPa. Let us consider the case of Σ^o first. Recall that a solution to ppGMPa has to provide a deterministic mapping from a separation fault F to one of the partitions P_i of F which is designated as live. Consider a run ρ without a fault of Σ^o where the quorums of all processes regardless of whether they are in P_i or not, intersect only on the processes inside P_i. Notice that in a run ρ' with fault F, the outputs of processes in the live partition do not have to change. Thus, Σ^o is not able to help an algorithm solve the ppGMPa.

Let us now consider Ω^o. This case is more complicated as for every fault, there is always a partition where every process has to change its leader: this is the partition that is separated from the leader of the whole system. Thus, presumably, a ppGMPa solution using Ω^o may designate such partition to be live. However, even if such mapping exists, the processes inside a partition may not be able to use it. Consider a fault F that separates the system into more than two partitions $F = \{P_i, P_j, P_k, \cdots\}$. Assume that P_i contains the leader of the system. That is, $min(P_i) = min(N)$. Thus, the outputs of Ω^o for processes in P_j and P_k change in case of F. However, only one partition can be designated as live. Let it be P_j. That is, P_k should not comply with the liveness properties of the GMP. Consider another fault $F' = \{P_k, N \setminus P_k\}$. In this case, P_k is the only partition where every process changes its output in case of fault. Thus, P_k should be designated as live for F'. However, processes in P_k are not able to differentiate between F and F' even with the help of Ω^o. Thus, the processes in P_k may not decide whether to act as in live partition or as in non-live. Thus, Ω^o is not able to help an algorithm solve the ppGMPa. Hence Conjecture 2.