Managed Agreement: Generalizing Two Fundamental Distributed Agreement Problems

Emmanuelle Anceaume, Roy Friedman, Maria Gradinariu

Managed Agreement: Generalizing Two Fundamental Distributed Agreement Problems

Emmanuelle Anceaume [a] Roy Friedman [b] Maria Gradinariu [c]

[a] *IRISA, Rennes, France, anceaume@irisa.fr*

[b] *Computer Science Department, The Technion, Haifa, Israel, roy@cs.technion.ac.il*

[c] *LIP6, Universite Paris 6, France, Maria.Gradinariu@lip6.fr*

Abstract

This paper presents a family of agreement problems called *Managed Agreement*, which is parameterized by the number of *aristocrat* nodes in the system; NBAC is a special case of this family when all nodes are aristocrats while Consensus is a special case of this family when there are no aristocrats. The paper also presents a parameterized family of failure detectors $\mathcal{F}(A)$ such that $\mathcal{F}(A)$ is the weakest failure detector class that enables solving Managed Agreement with a set A of aristocrats in an asynchronous environment.

1 Introduction

Consensus [12] and Non-Blocking Atomic Commit (NBAC) [2,13] are two fundamental distributed agreement problems. Intuitively, the specification of these problems assumes that each node in a distributed system starts with a given input value and the goal of each node is to decide on some output value. However, the decided values are restricted such that: all processes that do not crash eventually decide (*termination*), the value decided by all processes is the same (*agreement*), and the value decided must be related to the initial input values (*validity*). The difference between Consensus and NBAC is in their validity requirements. Specifically, Consensus only requires that a decided value is also a value that was proposed. In NBAC, it is assumed that the possible initial values are *yes* and *no* and the possible decision values are *commit* and *abort*. If the initial value of at least one node is *no*, then the decision must be *abort*. On the other hand, if the initial values of all nodes are *yes* and there are no crash failures, then the decision value must be *commit*.

Despite the similarity in structure of the definitions of Consensus and NBAC, in asynchronous distributed systems prone to crash failures, these are two different problems [10]. In particular, neither problem can be solved in a purely asynchronous system. However, it was shown that the minimal synchrony required to solve Consensus is strictly weaker than the one required to solve NBAC [5] (we make this statement more precise below). On the other hand, a black-box implementation of NBAC is not sufficient to solve Consensus in an otherwise asynchronous environment.

In this paper we propose a family of problems that we call *Managed Agreement* [1], which generalizes both NBAC and Consensus. Managed Agreement aims at forcing a decision via a set of special nodes. This approach is complementary to approaches that aim at speeding up the global decision by locally analyzing the set of proposed values [6]. Specifically, the definition of Managed Agreement is based on the notion of *aristocrat* nodes. In Managed Agreement, there exists a value such that if any of the aristocrats proposes this value, then a corresponding value must be decided. On the other hand, if none of the aristocrats proposed the special value and none of the aristocrats failed, then any possible decision value that corresponds to a value that was proposed can be decided on. Thus, NBAC is a special case of Managed Agreement when all nodes are aristocrats, whereas Consensus is a special case of Managed Agreement when there are no aristocrats.

In this paper we present a generic protocol for solving Managed Agreement, which is based on a transformation from Consensus to NBAC by Guerraoui [9]. The protocol we present utilizes any known Consensus protocol as a black-box and a new class of failure detector that we denote $?\mathcal{P}_{\mathrm{Ar}}(A)$, which is an extension of the known $?\mathcal{P}$ to detect crashes of aristocrats only ($?\mathcal{P}$ was introduced in [9]). Finally, we introduce a failure detector class $\Psi_{\mathrm{Ar}}(A)$, again, an extension of the known class Ψ [5], and show that a corresponding family of failure detectors, denoted $\mathcal{F}(A) = (?\mathcal{P}_{\mathrm{Ar}}(A), \Psi_{\mathrm{Ar}}(A))$, is the weakest class of failure detectors that enables solving Managed Agreement for a given A. [2]

2 Asynchronous Distributed Systems with Process Crashes

The computation model follows the one described in [4,7]. The system consists of a finite set Π of $n > 1$ processes, namely, $\Pi = \{p_1, \ldots, p_n\}$. A process can fail by *crashing*, i.e., by prematurely halting. At most $f < n$ processes can fail by crashing. A process behaves correctly (i.e., according to its specification) until it (possibly) crashes. By definition, a *correct* process is a process that does not crash. A *faulty* process is one that is not correct. Until it (possibly) crashes, a process is *alive*.

[1] The term Managed Agreement is inspired by the term “managed democracy”.

[2] Given two families of failure detectors D_1 and D_2, we denote by (D_1, D_2) the family of failure detectors whose output for a given failure pattern $\mathcal{T}$ can be any tuple (s_1, s_2) such that s_1 is a valid output for D_1 on $\mathcal{T}$ and s_2 is a valid output for D_2 on $\mathcal{T}$.

Processes communicate and synchronize by sending and receiving messages through channels. Every pair of processes is connected by a channel. Channels are assumed to be reliable. There is no assumption about the relative speed of processes nor on message transfer delays: the system is asynchronous.

We further enhance the environment, denoted $\mathcal{E}$, by assuming that each process has access to (one or more) *failure detector(s)* [4]. A failure detector is a module that provides each process with possibly inaccurate information about the occurrence of failures in the system. Below, we list four known types of failure detectors that enable solving Consensus and NBAC in an otherwise asynchronous distributed system, and then propose a new one, namely, $\mathcal{F}(\mathcal{A})$.

The class quorum failure detector Σ: Specifically, Σ outputs at each process a set of processes such that any two sets intersect, and eventually every set output at correct processes consists only of correct processes. It was shown in [5] that Σ is the weakest failure detector to implement atomic registers.

The class leader failure detector Ω: The failure detector Ω outputs the id of some process at each process. There is a time after which it outputs the id of the same correct process at all correct processes [3]. It was shown in [5] that (Ω, Σ) is the weakest failure detector to solve Consensus for all environments.

The class $?\mathcal{P}$: A failure detector that belongs to the class $?\mathcal{P}$ provides a boolean value to each process while maintaining the following property [9]:

- Anonymous Accuracy: The failure detector eventually returns true iff some process in the system has crashed.

The class Ψ: For an initial period of time, the output of Ψ at each process is *false*. Eventually Ψ behaves either like the failure detector (Ω, Σ) at all processes, or, in case a failure previously occurred, it may behave like the failure detector $?\mathcal{P}$ at all processes. The switch from *false* to $?\mathcal{P}$ is allowable only if a failure previously occurred. In [5] it is proved that $(\Psi, ?\mathcal{P})$ is the weakest failure detector to solve NBAC, while Ψ is the weakest failure detector to solve another intermediate problem called Quitable Consensus (this latter problem was also introduced in [5]).

3 Managed Agreement Problem

In the Managed Agreement problem, the set of possible proposed values, *P-VALS*, can be different from the set of possible decided values, *D-VALS*. However, we require a one-to-one mapping $\mathcal{M}$ from the set *P-VALS* to the set *D-VALS*. In particular, for each value $pv \in$ *P-VALS* and value $dv \in$ *D-VALS* such that $dv = \mathcal{M}(pv)$, we say that dv is the *value that corresponds to* pv. Moreover, we identify one special value in *P-VALS* as the *default value* and denote it *Default*. To clarify, *Default* is just a generic symbol; for example, in the case of NBAC, the value *no* is the *Default*. We also identify a set A of *aristocrats* among the entire set of nodes (this subset is a parameter). Managed agreement is then defined by the following properties:

- (Uniform) Agreement: No two processes decide differently.
- Termination: Every correct process eventually decides on some value.
- Managed-Obligation: If the decision value is $\mathcal{M}(\textit{Default})$, then either one of the aristocrats proposes *Default* or crashes.
- Managed-Justification: If the decision value v is different from $\mathcal{M}(\textit{Default})$, then v corresponds to a proposed value and all aristocrats propose a non default value.

Notice that a decision on a value other than $\mathcal{M}(\textit{Default})$ requires all aristocrats to propose a value different from *Default*. In particular, in runs in which at least one of the aristocrats have crashed even before the beginning of the protocol, the only possible decision value is $\mathcal{M}(\textit{Default})$.

4 The Weakest Failure Detector to Solve Managed Agreement

We introduce a novel class of failure detectors: the class $\mathcal{F}(\mathcal{A})$. First, we introduce the class of failure detectors $?\mathcal{P}_{\mathrm{Ar}}(A)$. Specifically, given a set of aristocrats A, a failure detector in $?\mathcal{P}_{\mathrm{Ar}}(A)$ provides a boolean value (initially *false*) to each process while maintaining the following property:

- Aristocratic Accuracy: The failure detector eventually returns true iff some process in A has crashed.

Second, we can similarly extend the definition of Ψ to $\Psi_{\mathrm{Ar}}(A)$ in the obvious way. That is, $\Psi_{\mathrm{Ar}}(A)$ initially outputs $\perp$ at all processes. Then it either behaves as (Ω, Σ), or if one of the aristocrats in A has crashed, it may behave as $?\mathcal{P}_{\mathrm{Ar}}(A)$. However, its behavior has to be consistent at all processes. Notice that when $A = \emptyset$, then $?\mathcal{P}_{\mathrm{Ar}}(A)$ always return *false* and $\Psi_{\mathrm{Ar}}(A)$ degenerates to (Ω, Σ). Finally, we define the class $\mathcal{F}(A) = (?\mathcal{P}_{\mathrm{Ar}}(A), \Psi_{\mathrm{Ar}}(A))$.

4.1 Sufficiency: Managed Agreement Through Consensus

A generic protocol for solving Managed Agreement based on the availability of a failure detector $\mathcal{F}(A)$ appears in Figure 1. Every aristocrat first sends its proposed value to all other processes. Every process then waits until either it has received the proposed values of all aristocrats, or the $?\mathcal{P}_{\mathrm{Ar}}(A)$ part of its $\mathcal{F}(A)$ failure detector told it that one of the aristocrats has crashed. The latter is used to avoid blocking

```
Function ManagedAgreement(A,$v_i, k_i$)
    % $k_i$ is a parameter aimed as distinguishing between different instantiations of the protocol
    if I am an aristocrat (in A) then
        send (VOTE,$v_i, k_i$) to everyone;
    endif;
    wait until either (VOTE,-,$k_i$) messages have been received from every aristocrat or $?\mathcal{P}_{\mathrm{Ar}}(A)$ returns true;
    if received a (VOTE,Default,$k_i$) message from at least one aristocrat or $?\mathcal{P}_{\mathrm{Ar}}(A)$ returned true then
        let $u_i := \mathcal{M}(Default)$
    else
        let $u_i := \mathcal{M}(v_i)$
    endif;
    while ($\Psi_{\mathrm{Ar}}(A)$ = $\perp$) do nop done
    if $\Psi_{\mathrm{Ar}}(A)$ = true then /* $\Psi_{\mathrm{Ar}}(A)$ behaves as $?P_{\mathrm{Ar}}(A)$
        return $\mathcal{M}(Default)$
    else
        $val$:=consensus($u_i, k_i$); /* $\Psi_{\mathrm{Ar}}(A)$ behaves as ($\Omega$,$\Sigma$)
        return $val$
    endif
```

Fig. 1. A Managed Agreement Protocol Based on $\mathcal{F}(A)$ and a Consensus Subroutine

forever in case one of the aristocrats has crashed before sending its value to all other processes. Then, if a node received at least one *Default* value, or detected one aristocrat failure, it starts a Consensus protocol by proposing the value that corresponds to *Default* in the Managed Agreement specification. Otherwise, it starts the Consensus with a value that corresponds to its own proposed value. Yet, before starting the Consensus protocol, the process must wait until the $\Psi_{\mathrm{Ar}}(A)$ failure detector makes up its mind on whether it behaves as $?\mathcal{P}_{\mathrm{Ar}}(A)$ or as (Ω, Σ). In the former case, this means that one of the aristocrats has failed, and this has been observed by all correct processes. Thus, it is safe to decide $\mathcal{M}(Default)$. Otherwise, the consensus is invoked, since it is known that the failure detector's output obeys (Ω, Σ), and thus the Consensus protocol will terminate correctly. In particular, the use of Consensus ensures agreement between all nodes while verifying that the agreed value also maintains validity. Let us note that when there are no aristocrats, this protocol trivially degenerates to invoking Consensus with the initial values.

Lemma 1 *The protocol in Figure 1 solves the Managed Agreement problem in asynchronous environments in which processes are equipped with a failure detector from the class $\mathcal{F}(A)$ and a Consensus subroutine.*

Proof: It is easy to see that the agreement property trivially holds. If a process decides $\mathcal{M}(Default)$ due to finding $\Psi_{\mathrm{Ar}}(A)$ = *true*, then by definition, all correct processes do the same and decide $\mathcal{M}(Default)$. Otherwise, if Consensus is invoked, then all processes decide the output of Consensus, and thus agreement follows from the correctness of the Consensus protocol. Similarly, the termination property holds due to the termination property of the Consensus protocol and the use of the $?\mathcal{P}_{\mathrm{Ar}}(A)$ part of $\mathcal{F}(A)$ in the **wait** statement. Thus, we only need to show validity. Clearly, if processes find $\Psi_{\mathrm{Ar}}(A)$ = *true*, then this means that one of the aristocrats has crashed. In this case, they all decide $\mathcal{M}(\textit{Default})$ and therefore validity is preserved. Therefore, for the rest of this proof we concentrate only on runs in which the $\Psi_{\mathrm{Ar}}(A)$ part of the failure detector acts like (Ω, Σ). Let us first consider runs of the protocol in which none of the aristocrats crashes. In these runs, at the end of the **wait** statement, every alive process has all the proposed values of all the aristocrats. Thus, if any of the aristocrats has proposed the default value, then all processes (that do not crash beforehand) start the Consensus with the corresponding value $\mathcal{M}(\textit{Default})$. By the validity of Consensus, the returned value by Consensus must also be $\mathcal{M}(\textit{Default})$. Therefore, in these cases Managed-Obligation is observed. Alternatively, if none of the aristocrats proposes *Default*, then every process (that does not crash beforehand) starts the Consensus with a value u_i that corresponds to its proposed value v_i. By the validity of Consensus, the decided value must be one of these u_i values. Consequently, Managed-Justification is observed in these runs.

The only thing left to show now is that in runs in which at least one aristocrat proposes *Default* and at least one aristocrat crashes (either the same aristocrat or a different one), then the decided value corresponds to the default value. When considering such a run, at the end of the **wait** statement, every alive process either has received at least one *Default* value, or has had its $?\mathcal{P}_{\mathrm{Ar}}(A)$ return *true*. In either case, the process starts Consensus with $\mathcal{M}(\textit{Default})$. By the validity of Consensus, this is also the decided value. ■

4.2 Necessity: The Minimal Failure Detector for Solving Managed-Agreement

In the following, we show that any failure detector $\mathcal{D}$ that enables solving Managed Agreement can be transformed into both $?\mathcal{P}_{\mathrm{Ar}}(A)$ and $\Psi_{\mathrm{Ar}}(A)$. Consequently, it implements $\mathcal{F}(A)$.

4.2.1 From Managed-Agreement to $?\mathcal{P}_{\mathrm{Ar}}(A)$

In the following, we show that any failure detector $\mathcal{D}$ that enables solving Managed Agreement in any environment can be transformed into $?\mathcal{P}_{\mathrm{Ar}}(A)$. The transformation algorithm that we use is similar to the one proposed in [9] for emulating $?\mathcal{P}$ from NBAC. The algorithm works as follows. Each process p_i has a local boolean variable $\textit{output}_{p_i}$, which provides the information that should be returned by its local failure detector $?\mathcal{P}_{\mathrm{Ar}(A)}$. We assume the existence of the function

`ManagedAgreement()` that solves the Managed Agreement problem. Each process p_i initiates *output*$_{p_i}$ to *false* and then repeatedly invokes the Managed Agreement function with a non default value. This is done forever, unless the returned value is $\mathcal{M}(\textit{Default})$, in which case p_i changes *output*$_{p_i}$ to *true* and exits. The idea is that by the definition of Managed Agreement, $\mathcal{M}(\textit{Default})$ can only be returned in this case if and only if at least one of the aristocrats has failed. The exact pseudocode appears in Figure 2 and the proof is given below.

```
$\textit{output}_{p_i}$ := $\textit{false}$; $k_i$ := 1;
select $v_i \in \textit{P-VALS} \setminus \{\textit{Default}\}$
repeat
    $t_i$ := ManagedAgreement($A, v_i, k_i$); $k_i$ := $k_i + 1$;
until $t_i = \mathcal{M}(\textit{Default})$;
$\textit{output}_{p_i}$ := $\textit{true}$;
```

Fig. 2. From Managed Agreement to $?\mathcal{P}_{\mathrm{Ar}}(A)$

Lemma 2 *The transformation algorithm in Figure 2 emulates the failure detector* $?\mathcal{P}_{\mathrm{Ar}(A)}$.

Proof: We prove that the transformation algorithm verifies the Aristocratic Accuracy property, i.e., it eventually returns *true* if and only if some process in A has crashed. Suppose the transformation algorithm outputs *true* at some point. This can happen only if some invocation of the `ManagedAgreement()` function returns $\mathcal{M}(\textit{Default})$. By definition, this can happen either if one of the aristocrats proposes *Default* or crashes. The first scenario is impossible since all processes invoke `ManagedAgreement()` with a non *Default* value. Therefore, the output can be *true* only due to an aristocrat's crash.

Let us consider now an execution of the protocol where an aristocrat crashes, and assume w.l.o.g. that it crashes before invoking the kth instance of `Managed Agreement()`. Also, let p_i be a correct process executing the transformation algorithm. By the Termination property, the kth invocation by p_i of `Managed Agreement()` in the transformation algorithm eventually returns; by the Managed-Obligation property, the returned value in this case must be $\mathcal{M}(\textit{Default})$. Therefore, the transformation algorithm terminates by setting the output to *true*. ■

4.2.2 From Managed Agreement to $\Psi_{\mathrm{Ar}}(A)$

The transformation from Managed Agreement to $\Psi_{\mathrm{Ar}}(A)$ is inspired by the transformation that was first proposed in [3], and then in [5], to extract Ψ from any failure detector $\mathcal{D}$ that solves Quitable Consensus. Specifically, let $\mathcal{D}$ be an arbitrary failure detector that can be used to solve Managed Agreement in some environment $\mathcal{E}$. Let $\mathcal{A}lg$ be an algorithm that uses $\mathcal{D}$ to solve Managed Agreement in $\mathcal{E}$. We must prove that $\Psi_{\mathrm{Ar}}(A)$ can be "extracted" from $\mathcal{D}$ in environment $\mathcal{E}$, i.e., processes can run in $\mathcal{E}$ a transformation algorithm that uses $\mathcal{D}$ and $\mathcal{A}lg$ to generate the output of $\Psi_{\mathrm{Ar}}(A)$— a failure detector that initially outputs $\perp$ and later behaves either like (Ω, Σ) or like $?\mathcal{P}_{\mathrm{Ar}(A)}$. The reduction algorithm $T_{\mathcal{D} \rightarrow \Psi_{\mathrm{Ar}}(A)}$ is shown in Figure 3.

The basic idea of the transformation is to have each process locally simulate the behavior of the overall distributed system. That is, to simulate the execution (by all the processes of the system) of runs of algorithm $\mathcal{A}lg$ that could have occurred in the current failure detector pattern F and failure detector history of $\mathcal{D}$ (this is done by Task 1 in Figure 3). Additionally, determine whether in the current run of the simulation it is possible to extract (Ω, Σ), or it is legitimate to start behaving like $?\mathcal{P}_{\mathrm{Ar}(A)}$ and output true because one of the aristocrats has crashed (see Task 2 in Figure 3).

Notice that since we assume that $\mathcal{A}lg$ solves Managed Agreement, we can assume, w.l.o.g., that it solves this problem when $\{0, 1, \textit{Default}\} \in$ *P-VALS* and $\mathcal{M}(i) = i$ for $i \in \{0, 1\}$. With this observation, we detail below both tasks.

In the construction, each process p starts by outputting $\perp$. In Task 1, p simulates runs of $\mathcal{A}lg$ that could have occurred in the current failure pattern F and the current failure detector history of $\mathcal{D}$, exactly as in [3] (see below for extended definitions). It does this by "sampling" its local failure detector $\mathcal{D}$ and exchanging failure detector samples with the other processes (Line 5 in Figure 3). Process p organizes these samples into ever-increasing DAG G_p whose edges are consistent with the order in which the failure detectors samples were taken. Using G_p, p simulates ever-increasing partial runs of $\mathcal{A}lg$ that are compatible with paths in G_p. A path from the root of a tree to a node x in the tree corresponds to the schedule of a partial run of the algorithm, where every edge along the path corresponds to a step of some process. Each process p organizes these runs in a forest induced by $T = \begin{pmatrix} n+p-1 \\ n \end{pmatrix}$ configurations, with n the number of processes, and p the number of different input values, i.e., $p = 3$. This forest, denoted Υ_p, contains T trees. We can order these configurations such that configurations I^i and I^{i-1}, with $0 \leq i < T$ differ only in the value of one proposition. These trees are ordered such that Υ_p^0 corresponds to simulated runs of $\mathcal{A}lg$ in which all the processes propose 0, Υ_p^k with some $k > 0$ corresponds to simulated runs of $\mathcal{A}lg$ in which all the processes propose 1, and Υ_p^{T-1} in which all the processes propose *Default*. Note that it exists an ordering which guarantees that there is no k' with $0 \leq k' \leq k$ such that some process proposes *Default* in $\Upsilon_p^{k'}$.

Processes periodically query their failure detectors. The results of these queries include failure and temporal information. Each process exchanges the results of its queries with all the other processes. Upon receipt of such information, a process construct a DAG [3] by incorporating the received information to its own DAG. (Each process exchanges its whole DAG with all the other processes. The temporal information enables to incorporate the received DAG with the local one.) Thus every (correct) process can construct ever-increasing finite approximations of the same infinite limit DAG G. This DAG is then used to simulate runs of managed agreement. Specifically, each path g within G can be used to simulate schedules of runs of managed agreement. That is, a path g represents several possible schedules and failure detectors values for the processes during their execution of managed agreement. There are many different schedules that match a path in DAG G because each schedule depends on the order in which messages are received. Thus, if we consider each initial configuration I^i then one can construct a tree rooted at I^i. The set of vertices of the tree rooted at I^i is the set of all possible schedules that can occur from the given configuration I^i. An edge corresponds to an event "receipt by a process p of a message m, and the failure detector value seen by the sender of the message when it sent m to p". By considering the T different configurations, one obtains a forest of simulated runs of managed agreement. Thus the infinite limit DAG G induces an *infinite limit forest,* Υ. The *limit tree* of Υ_p^i is denoted Υ^i. Each node S of Υ is tagged by the set of decisions that correct processes reach in the partial runs that are the descendants of S. These tags can be either univalent, i.e., 0-valent or 1-valent or $\mathcal{M}(\textit{Default})$-valent, or multi-valent (i.e., with more than one tag). We use the same definition for a critical index as in [5]: Index $i \in \{0, \ldots, T-1\}$ is critical if the root of Υ^i is multivalent or the root of Υ^i is u-valent and the root of Υ^{i-1} is v-valent, with $u, v \in \{0, 1, \mathcal{M}(\textit{Default})\}$ and $u \neq v$.

In Task 2, p waits until it decides in some simulated run of every tree of the forest Υ_p (Line 8 in Figure 3). If p decides $\mathcal{M}(\textit{Default})$ in any of these runs and the initial configuration of this run does not contain any *Default* value proposed by an aristocrat then a failure must have occurred (in the current failure pattern). Note that this condition is stronger than the one in [5] because of the Managed Obligation property of Managed Agreement. Thus p knows that it is legitimate to propose the extraction of $?\mathcal{P}_{\mathrm{Ar}}(A)$. Otherwise, p's decision values in the simulated runs are 0s, 1s or $\mathcal{M}(\textit{Default})$ but in this latter case, the initial configuration contains the *Default* value (proposed by an aristocrat), and thus does not tell anything regarding failures. Thus p determines that it is possible to extract (Ω, Σ). Note that by the validity properties of Managed Agreement, starting from an initial configuration in which there is no *Default* values (proposed by an aristocrat), the decision value may be either $\mathcal{M}(\textit{Default})$ if an aristocrat has failed after having voted and this failure has been detected before the receipt of his vote, or a decision value $v \neq \mathcal{M}(\textit{Default})$, otherwise. At this point, p executes the given Managed Agreement algorithm $\mathcal{A}lg$ to agree with all the processes on whether to extract $?\mathcal{P}_{\mathrm{Ar}}(A)$ (because at least p has detected an aristocrat failure) or to extract (Ω, Σ). Specifically, in the former case, process p invokes an instance of $\mathcal{A}lg$ by proposing *Default.* In the latter case, it invokes $\mathcal{A}lg$ with (I, I', S, S') value, where I and I' are initial configurations that differ only in the proposal of one process while S and S' are schedules in the simulated forest so that processes decide u in $S(I)$ and v in $S'(I')$, where $u, v \in \{0, 1\}$ and $u \neq v$. The proof of existence of such configurations and schedules is similar to the one shown in Lemma 2 in [8], and detailed in [1].

If processes decide to extract (Ω,Σ), they continue the simulation of runs of $\mathcal{A}lg$ to do this extraction. Note that this extraction cannot start if the decision value in every simulated run is $\mathcal{M}(\textit{Default})$. Notice that the failure of an aristocrat may not be detected. Hence the necessity of the algorithm shown in Figure 2 that emulates $?\mathcal{P}_{\mathrm{Ar}}(A)$ with Managed Agreement. If $\mathcal{A}lg$ returns $\mathcal{M}(\textit{Default})$, then $\mathcal{A}lg$ starts to behave like $?\mathcal{P}_{\mathrm{Ar}}(A)$: p stops outputting $\perp$ and outputs *true* from that time on (Line 16). If $\mathcal{A}lg$ returns a value of the form (I, I_0, S, S_0), then p stops outputting $\perp$ and starts extracting Ω (Line 21) and Σ (Line 23). The extraction of Ω is done using the procedures of both [3] and [5]. To extract Ω, p must continuously output the identifier of a process such that eventually, correct processes output the identifier of the same correct process. The existence of a correct process relies on the existence of a critical index (see Lemma 3 below). Finally, the extraction of Σ is done exactly as in [5] and detailed in [11].

Lemma 3 *If any process reaches Line 21, then the limit forest Υ has a critical index.*

Proof: (Adaptation of Lemma 3 in [5]). If a process reaches Line 21, then it must have previously decided a tuple (I_0, I_1, S_0, S_1). By the Managed-Justification of Managed Agreement, some process q must have proposed this tuple to algorithm $\mathcal{A}lg$. Since q proposed this tuple, it must have decided some value v different from $\mathcal{M}(\textit{Default})$ in some run of Υ_q (this follows from the choice of tuple (I_0, I_1, S_0, S_1)). By construction of the limit forest, all the correct processes are aware of the partial run that allowed q to decide value v, and include this partial run to their own forest. By Termination and Agreement of Managed Agreement, all the correct processes decide $v \neq \mathcal{M}(\textit{Default})$ in some run of Υ. From above, the root of some tree Υ^i is tagged with $v \neq \mathcal{M}(\textit{Default})$. Two cases are possible. Either the root is uni-valent, i.e., it is tagged with only value v, or it is multi-valent, i.e., it is tagged with both v and other tags ($1 - v$, and/or $\mathcal{M}(\textit{Default})$). In the latter case, we are done by the definition of a critical index. In the former case, all the roots are uni-valent. We consider two cases: Suppose first that $v = 1$. Two sub-cases are possible. Either *a)* $i \leq k$ (recall that index k corresponds to the tree in which all the processes propose value 1), or *b)* $i > k$. In sub-case *a)*, by considering the sequence $\Upsilon^0, \ldots, \Upsilon^i, \ldots, \Upsilon^k$, there must exist some index k' with $0 < k' \leq i$ such that the root of $\Upsilon^{k'-1}$ is 0-valent while the root of $\Upsilon^{k'}$ is 1-valent. By definition, k' is a critical index. In sub-case *b)*, by considering the sequence $\Upsilon^k, \ldots, \Upsilon^i, \ldots, \Upsilon^{T-1}$, there must exist some index k'' with $i < k'' \leq T - 1$ such that the root of $\Upsilon^{k''-1}$ is 1-valent while the root of $\Upsilon^{k''}$ is u-valent, with $u \in \{0, \mathcal{M}(\textit{Default})\}$. By definition k'' is a critical index. The case for $v = 0$ is similar to the case $v = 1$. ■

Theorem 1 *For all environments $\mathcal{E}$, if failure detector $\mathcal{D}$ can be used to solve Managed Agreement in $\mathcal{E}$, then the algorithm shown in Figure 3 transforms $\mathcal{D}$ into $\Psi_{\mathrm{Ar}}(A)$ in $\mathcal{E}$.*

Proof: The proof follows the lines of Theorem 6 in [5]. The only difference concerns the validity property. Specifically, for each process p, if $\Psi_{\mathrm{Ar}}(A) - output_p$ is true then it must be the case that some aristocrat has crashed during the current run. Suppose that process p outputs true in Line 16 in Figure 3, then p decided $\mathcal{M}(\mathit{Default})$ in the current execution of algorithm $\mathcal{A}lg$ (see Line 15). Thus, by Managed-Obligation of Managed Agreement, it must be the case that some process q invoked $\mathcal{A}lg$ with value *Default* as initial proposition (see Line 10) or that some aristocrat crashed during the current execution of $\mathcal{A}lg$. In the latter case, we are done. In the former case, q invoked $\mathcal{A}lg$ with value *Default* only if q decided $\mathcal{M}(\mathit{Default})$ in one of the simulated runs of $\mathcal{A}lg$ and if the initial configuration of this run did not contain any *Default* value proposed by some aristocrat. Thus, by Managed-Obligation of Managed Agreement, it must be the case that some aristocrat crashed during this run. The rest of the proof of this theorem is exactly the same as in [5]. ■

Theorem 2 $\mathcal{F}(A)$ *is the weakest class of failure detectors that enables solving Managed Agreement.*

Proof: The theorem follows directly from Lemma 1, Lemma 2 and Theorem 1. ■

References

[1] E. Anceaume, R. Friedman, and M. Gradinariu. Managed agreement: Generalizing two fundamental distributed agreement problems. Technical Report 1785, IRISA, 2006.

[2] P. Bernstein, V. Hadzilacos, and H. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.

[3] T. Chandra, V. Hadzilacos, and S. Toueg. The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[4] T. Chandra and S. Toueg. Unreliable Failure Detectors for Asynchronous Systems. *Journal of the ACM*, 43(4):685–722, 1996.

[5] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, and S. Toueg. The Weakest Failure Detectors to Solve Certain Fundamental Problems in Distributed Computing. In *Proc. 23rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 338–346, 2004.

[6] P. Dutta, R. Guerraoui, and B. Pochon. Fast non-blocking atomic commit: An inherent trade-off. *Information Processing Letters*, 91(4):195–200, 2004.

[7] M. Fischer, N. Lynch, and M. Patterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, 1985.

[8] R. Guerraoui, V. Hadzilacos, P. Kouznetsov, and S. Toueg. The weakest failure detectors to solve quitable consensus and non-blocking atomic commit. Technical report, LPD EPFL, 2004.

[9] Rachid Guerraoui. Non-Blocking Atomic Commit in Asynchronous Distributed Systems with Failure Detectors. *Distributed Computing*, 15(1):17–25, 2002.

[10] V. Hadzilacos. On the Relationship Between the Atomic Commitment and Consensus Problems. *Fault-Tolerant Distributed Computing*, pages 201–208, 1990.

[11] P. Kouznetsov. *Synchronizations using Failure Detectors*. PhD thesis, School of Computer and Communication Sciences, EPFL, Suisse, 2005.

[12] L. Pease, P. Shostak, and L. Lamport. Reaching Agreement in Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980.

[13] D. Skeen. Crash Recovery in a Distributed Database System. Memorandum No. UCB/ERL M82/45, Electronics Research Laboratory, Berkeley, 1982.

```
Initially:  (1) $T := \binom{n+p-1}{n}$
            (2) $\Psi_{\mathrm{Ar}}(A) - output_p := \bot$ {$\Psi_{\mathrm{Ar}}(A) - output_p$ is the output of module
                $\Psi_{\mathrm{Ar}}(A)$ at $p$ }
Task 1:
 (3)  do forever /* same construction as in [3] */
 (4)  cobegin /* cases a) and b) are executed in parallel */
 (5)    a) $p$ builds an ever-increasing DAG $G_p$ of failure detectors samples by repeatedly
           sampling its failure detector $\mathcal{D}$ and exchanging these samples with the processes
 (6)    b) $p$ uses $G_p$ and the $T$ initial configurations to construct a forest $\Upsilon_p$ of ever-
           increasing simulated runs of $\mathcal{A}lg$ using $\mathcal{D}$ that could have occurred with the
           current failure pattern $F$ and the current failure detector history.
 (7)  coend
Task 2:
 (8)  wait until ($\forall\ \Upsilon_p^i$, with $0 \leq i \leq (T-1)$, $p$ decides in one of the runs of $\Upsilon_p^i$)
 (9)  if ($\exists\ i$ such that $p$ decides $\mathcal{M}(\textit{Default})$ in $\Upsilon_p^i$ and $I^i$ does not contain any Default
      value proposed by an aristocrat) then
 (10)   $p$ runs $\mathcal{A}lg$ with Default as input value
 (11) else /* $\forall\ \Upsilon_p^i$, $\exists$ a run in which the decision value is either 0, 1, or $\mathcal{M}(\textit{Default})$ but
      in the latter case, $\textit{Default} \in I^i$ */
 (12)   select two configurations $I^{i-1}$ and $I^i$, with $1 \leq i \leq (T-1)$ and two schedules $S$
        and $S'$ st $p$ decides $u$ in $S(I^{i-1})$ and $v$ in $S'(I^i)$, with $u, v \in \{0, 1\}$ and $u \neq v$
 (13)   $p$ runs $\mathcal{A}lg$ with $(I, I', S, S')$ as input value
 (14)   wait until ($p$ decides in $\mathcal{A}lg$)
 (15)   if ($p$ decides $\mathcal{M}(\textit{Default})$) then
 (16)     $\Psi_{\mathrm{Ar}}(A) - output_p := true$ /* $\Psi_{\mathrm{Ar}}(A)$ behaves as $?\mathcal{P}_{\mathrm{Ar}(A)}$ */
 (17)   else /* $p$ decides $(I_0, I_1, S_0, S_1)$ */
 (18)     $\Omega - output_p := p$
 (19)     $\Sigma - output_p := \Pi$
 (20)     cobegin /* cases a), b) and c) are executed in parallel */
 (21)       a) do forever /* extractions of $\Omega$ */
 (22)         $\Omega - output_p \leftarrow q$ such that $p$ extracts $q$ following the procedure in [3]
 (23)       b) let $\mathcal{C}$ be the set of configurations reached by applying all prefixes of $S_0$ to
            $I_0$ and $S_1$ to $I_1$
 (24)         do forever /* extraction of $\Sigma$ */
 (25)         $\Sigma - output_p \leftarrow \bigcup_{C \in \mathcal{C}}$ set of processes that $p$ extracts following the
              procedure in [5]
 (26)       c) do forever
 (27)         $\Psi_{\mathrm{Ar}}(A) - output_p := (\Omega - output_p, \Sigma - output_p)$
 (28)     coend
 (29)   endif
 (30) endif
```

Fig. 3. Extraction of $\Psi_{\mathrm{Ar}}(A)$ from $\mathcal{D}$ and Managed Agreement algorithm $\mathcal{A}lg$ – code for process p