Validity Conditions in Agreement Problems and Time Complexity

Bernadette Charron-Bost[1] and Fabrice Le Fessant[2]

[1] STIX, École Polytechnique, 91128 Palaiseau Cedex, France
[2] Microsoft Research Lab, 7 JJ Thomson Avenue, CB3 0FD, Cambridge, UK

Abstract. We first introduce a new class of distributed agreement problems, ranging from Uniform Consensus to Non-Blocking Atomic Commitment, by varying the validity condition in the specification. We then provide an early deciding algorithm to solve each problem of this class in the synchronous model with crash failures. Our algorithm achieves the previously established lower bounds for time complexity showing that these lower bounds are tight.

Keywords: Distributed Algorithms, Complexity.

1 Introduction

Reaching agreement in a distributed system is a fundamental problem which arises in numerous guises. In all versions of agreement problems, each process starts with an initial value from some fixed set V. At some point in the computation, correct processes must irrevocably decide on some values in V. The specification of an agreement problem includes two safety conditions which fundamentally depend on the failure model that is considered [10]. In this paper, we suppose that processes can fail only by crashing; for this failure model, all the agreement problems share the same safety condition, called *agreement*, which specifies that no two processes may decide differently. Another safety condition, called *validity*, describes the decision values that are permitted. The validity condition varies according to the particular version of the agreement problem. For instance, Fischer, Lynch, and Paterson [5] consider a weak form of validity, called *non-triviality*, which only specifies that there exist at least two runs with two different decision values. For the crash failure model, the validity condition of the *Consensus* problem, which we call *integrity*, requires every process's decision value to be the inital value of some process. Integrity is inappropriate as a validity condition for the transaction commitment problem where the initial votes of all processes influence the decision. Indeed, the validity condition of the *non-blocking atomic commitment* (*NB-AC*, for short) is *unanimity*: decide 1 (`commit`) only if every process's initial value is 1 (`yes`), and decide 0 (`abort`) if some process starts with 0 (`no`) or a failure occurs.

In this paper, we investigate the relationships between different agreement problems. Indeed, slight modifications in the validity condition may yield considerable discrepancy between agreement problems. For instance, Dwork and Moses [4] show that, in synchronous systems, two rounds are always sufficient for solving the agreement problem of [5] with the non-triviality condition, while any Consensus algorithm tolerating t failures requires $t+1$ rounds. In asynchronous systems Charron-Bost and Toueg [2] show that Consensus and NB-AC are incomparable in almost all environments: NB-AC is never reducible to Consensus, and Consensus is not reducible to NB-AC, except when at most one process may fail. However, we observe that increasing the synchrony degree allows us to compare these two problems, and makes them closer and closer: In each of the partially synchronous models described in [3], NB-AC is harder than Consensus; indeed if a majority of processes is correct, then Consensus is solvable but not NB-AC. Moreover, in synchronous systems Consensus and NB-AC are both solvable no matter how many processes fail, and so are equivalent in terms of solvability. Actually they are equivalent in a stronger sense: in the presence of up to t failures, $t+1$ rounds are necessary and sufficient in the worst case execution to solve as well Consensus as NB-AC [10]. One can refine this comparison by discriminating runs according to the number of failures f that actually occur. Charron-Bost and Schiper [1] and subsequently Keidar and Rajsbaum [6] proved that both Consensus and NB-AC require at least $f+2$ rounds if $f < t-1$, and only $f+1$ rounds if $t-1 \leq f \leq t$ (where t is the maximum number of failures).

One may wonder whether these lower bounds for early deciding in Consensus and NB-AC algorithms are tight. An algorithm for Consensus is presented in [1] that achieves these lower bounds. Unfortunately, this algorithm cannot be easily adapted for the NB-AC problem. As noticed by Lamport in [8], "a [popular] belief is that a three-phase commit protocol *à la* Skeen [12] is needed" for solving NB-AC. It is not exactly clear what that means, but it seems to imply that at least three rounds are required in failure-free runs for deciding. The $f+2$ lower bound would be therefore weak in the case of NB-AC.

In this paper, we show that this intuition is incorrect: we devise an algorithm for NB-AC that achieves the general lower bounds for early deciding of [1]. This proves that even when refining the comparison in this way (i.e., considering the actual number of failures f instead of the worst case t), Consensus and NB-AC remain two equivalent problems.

Another contribution of the paper is to introduce a natural extension of the NB-AC problem, which we call the *k-Threshold Agreement* problem. It turns out that this problem is also a generalization of Consensus. In this way, we connect Consensus and NB-AC from the strict view point of problem specification. Actually, our algorithm for NB-AC is general enough to work for any k-Threshold Agreement problem. Therefore all these agreement problems for various values of k, which can be proved incomparable in asynchronous systems [2], are equivalent in synchronous systems in the strong sense that a common optimal algorithm solves them.

2 Preliminaries

2.1 The model

In this paper, we consider the synchronous model for distributed systems described in [10], which we review briefly.

Let Π be a fully connected graph with n nodes. We consider some fixed message alphabet M, and we let `null` be a placeholder indicating the absence of message.

A *synchronous algorithm* A is a collection of n *processes*, one for each node of Π. Each process $p_i \in \Pi$, is defined by four components: a set of states $states_i$, a subset $start_i \subseteq states_i$ of initial states, a message-generation function $msgs_i$ mapping $states_i \times \Pi$ to a unique (possibly `null`) message, and a state transition function $trans_i$ mapping $states_i$ and vectors of messages indexed by Π to $states_i$.

An execution of the algorithm A begins with each process in an arbitrary initial state. Then each process p_i, in lock-step, repeatedly performs the following three steps: (1) apply $msgs_i$ to the current state to determine the messages to be sent and send these messages, (2) receive *all* the messages which are sent to p_i during the previous step, and (3) apply $trans_i$ to the current state and the just received messages to obtain the new state. The combination of these three steps is called a *round* of A.

A *run* of A models an execution of the entire system: it is defined by the collection of the initial states of processes, and the infinite sequence of rounds of A that are executed. The *time complexity* of a synchronous algorithm is measured in terms of the number of rounds until all the required outputs are produced.

We assume that links are reliable but we allow processes to fail by crashing, i.e., by stopping somewhere in the middle of their execution. A process may crash before or during some instance of the steps described above. In particular, a process may succeed in sending only a subset of the messages specified to be sent. After crashing at a round, a process does not send any message during subsequent rounds. A process is said to be *correct* in a run if it does not crash; otherwise it is said to be *faulty*. In the following, we assume that at least one process is correct.

2.2 Agreement Problems

We now focus on the two well-known agreement problems: Consensus and Non-Blocking Atomic Commitment. The *Consensus* problem considers a fixed collection of processes which have each an initial value drawn from some domain V. Processes must eventually decide on the same value; moreover, the decision value must be the initial value of some process. The *Non-Blocking Atomic Commitment (NB-AC)* problem arises in distributed database systems to ensure the consistent termination of transactions. Each process that participates in a database transaction arrives at an initial opinion about whether to commit the transaction or abort it, and votes `yes` or `no` accordingly. Processes must eventually reach a common decision (`commit` or `abort`). The decision to commit may be reached only if all processes initially vote `yes`. In this case, `commit` must be decided if there is no failure.

When identifying `yes` and `commit` with 1, and `no` and `abort` with 0, it appears that binary Consensus ($V = \{0, 1\}$) and NB-AC are two very similar problems. They share the following two conditions:

Agreement: No two processes (whether correct or faulty) decide differently.
Termination: All correct processes eventually decide.

Binary Consensus and NB-AC only differ in their *validity conditions*, i.e., the conditions describing the decision values that are permitted: in the Consensus problem, the possible decision values are related only to the initial values, contrary to the NB-AC problem for which the decision values depend on both input values *and* failure pattern. More precisely, the validity condition of Consensus is as follows:

Integrity: Any decision value for any process is the initial value of some process.

Note that, for binary Consensus, Integrity can be equivalently formulated as follows:

Binary Integrity: If all processes start with the same value v, then v is the only possible decision value.

The validity condition of NB-AC is:

Unanimity:
1. If any process starts with 0, then 0 is the only possible decision value.
2. If all processes start with 1 and there is no failure, then 1 is the only possible decision value.

At this point, it is relevant to consider the *weak validity condition* introduced in [7]:

Weak Validity: If there is no failure, then any decision value is the initial value of some process.

Obviously, the weak validity condition is weaker than the validity conditions of both Consensus and Atomic Commitment, but it is stronger than the one in [5] which only precludes the trivial protocols that always decide the same value. This latter condition, that we call *non-triviality*, is as follows:

Non-Triviality: For each value $v \in \{0, 1\}$, there is an execution in which some process decides v.

Binary Consensus and NB-AC specifications are clearly not comparable because of their incomparable validity conditions.[1] However, it is possible to link up these two problems from a syntactic standpoint. For that, we introduce the *k-Threshold Agreement* problem (the *k-TAg* problem, for short) that is parametrized by some integer $k \in \{1, \cdots, n\}$. The termination and agreement conditions of k-TAg are the same as those of Consensus and NB-AC, and its validity condition is as follows:

k-Validity:
1. If at least k processes start with 0, then 0 is the only possible decision value.
2. If all processes start with 1 and at most $k-1$ failures occur, then 1 is the only possible decision value.

This new problem is a natural generalization of the Atomic Commitment problem (1-TAg is exactly NB-AC), but turns out to be also a generalization of Consensus; Indeed, n-TAg actually corresponds to binary Consensus, since at most $n-1$ processes are faulty. We have thereby defined a chain of problems which interpolates between NB-AC and binary Consensus. The original motivation for this generalization is purely theoretical: it is interesting to connect two incomparable problems by exploiting differences in validity conditions. But it is easy to imagine actual situations in which such a generalization could be natural: for example, it might be desirable for processes to enforce them to decide `abort` only if a majority of processes initially vote `no`, but to require that they commit if all of them initially vote `yes` as soon as a minority of processes are faulty.

2.3 Lower Bounds for Time Complexity

A fundamental result in synchronous systems [11] is that if $n \geq t+2$, then any Consensus algorithm that tolerates t crashes must run $t+1$ rounds in some execution before all processes have decided. As noticed by Lynch [10], the proof of this result still works if one weakens the validity condition to the weak validity condition. So the $t+1$ lower bound also holds for NB-AC and more generally, for all the k-TAg problems. However, it no longer holds if one weakens validity to the non-triviality condition. Indeed, Dwork and Moses [4] devised a simple algorithm achieving non-triviality, agreement, and termination in only two rounds.

These remarks, which concern the generalization of the worst case time complexity of Consensus to other agreement problems, also apply when one refines the analysis of time complexity by discriminating runs according to the number of failures that *actually* occurs. More precisely, Charron-Bost and Schiper [1] show the following lower bounds:

[1] Furthermore, Charron-Bost and Toueg [2] showed that these two problems are also incomparable from an algorithmic point of view: NB-AC is not reducible to binary Consensus, and *vice-versa* (except when the maximum number of failures is 1).

Theorem 1. *Let A be a Consensus algorithm which tolerates t process crashes. For each f, $0 \leq f \leq t$, there exists a run of A with at most f crashes in which at least one process decides not earlier than during round $f+2$ if $f \leq t-2$, and not earlier than during round $f+1$ otherwise.*

Afterwards, Keidar and Rajsbaum [6] showed the same lower bounds by an alternative method. The two proofs use different techniques but both clearly extend to the weak validity condition. As a consequence, this shows that each k-TAg problem, and in particular NB-AC, requires at least $f+2$ rounds to decide if $0 \leq f \leq t-2$ and $f+1$ rounds if $t-1 \leq f \leq t$.

---

states$_\mathbf{i}$
    $rounds \in N$, initially 0
    $W \in (V \cup \{\perp\})^n$, initially $(\perp, \cdots, v, \cdots, \perp)$ where v is p_i's initial value
    $ready$, $halt$, Booleans, initially `false`
    $Rec\_Failed, Failed \subseteq \Pi$, initially $\emptyset$
    $decision \in V \cup \{$`unknown`$\}$, initially `unknown`

msgs$_\mathbf{i}$
    if $\neg halt$ then
        if $\neg ready$ then send $W[i]$ to all processes
        else send $(ready, W[i])$ to all processes

trans$_\mathbf{i}$
    if $\neg halt$ then
        $rounds := rounds + 1$
        let w_j be the value sent by p_j, for each p_j from which a message arrives
        if a message has arrived from p_j then $W[j] := w_j$
        else $W[j] := \perp$
        if $ready$ then
            if $rounds \leq t$ then $decision := W[i]$
            $halt :=$ `true`
        else if some message $(ready, w)$ arrives then
                $W[i] := w$
                $ready :=$ `true`
            else $Rec\_Failed := Failed$
                $Failed := \{p_j :$ no message arrives from p_j in the current round$\}$
                $W[i] := \Phi(rounds, W)$
                if $Rec\_Failed = Failed$ then $ready :=$ `true`
            if $ready$ and $rounds \geq t$ then $decision := W[i]$

Fig. 1. A General Algorithm for Early Deciding

---

3 A General Early Deciding Algorithm

We now describe a general algorithm for the k-TAg problem that tolerates t crashes and achieves the lower bounds of Theorem 1 for early deciding.

Our algorithm is based on a strategy borrowed from [9] for determining the time of decisions: each process p_i maintains a variable $Failed$ containing the set of processes that p_i detects to have crashed. During a round, p_i learns that p_j has crashed if p_i receives no message from p_j in this round. At the end of every round, each process p_i updates its variable $Failed$. If $Failed$ remains unchanged during some round, namely if p_i detects no new crash failure, then p_i becomes ready to decide. Any process which becomes ready at round r is permitted to decide at round $r+1$ if $r \leq t-1$ or as early as round r if $r = t$ or $r = t+1$; furthermore, it broadcasts a *ready* message at round $r+1$ both to inform other processes that it is ready to decide and to force the other processes to become ready too.

To determine its decision value, each process maintains a vector W of size n. Initially, every process sets its own component of W to its initial value, and the other components to $\perp$. At each round $r \geq 1$, every process p_i broadcasts its own component $W[i]$. Then it collects all the values it has just received into W; if p_i does not receive a value from p_j, it simply sets $W[j] := \perp$. Finally, p_i sets its own component to a new value which depends on the round number r and the values p_i has received, that is $W[i] = \Phi(r, W)$ where Φ is an arbitrary function [3] mapping $\{1, \cdots, t+1\} \times (V \cup \{\perp\})^n$ to V. A simple induction on the round number r shows that no process will ever compute the value of Φ on elements of $\{1, \cdots, t+1\} \times (\{\perp\})^n$. The formal code of the algorithm is given in Figure 1.

In this section, we first prove that if all the processes invoke the same function Φ to adjust their own components in W, then agreement is guaranteed. This common function is then determined according to the validity condition to satisfy: in particular, we specify a function Φ that guarantees the k-validity property.

3.1 Correctness Proof

Theorem 2. *For any mapping $\Phi : \{1, \cdots, t+1\} \times (V \cup \{\perp\})^n \to V$, the algorithm in Figure 1 satisfies the termination and agreement properties in a synchronous system with at most t faulty processes. Moreover, the algorithm achieves the lower bounds of Theorem 1 for early deciding.*

Proof: To prove this theorem, we use the following notation and terminology. The value of any variable x at process p_i just at the end of round r is denoted by $x_i^{(r)}$. By extension, $x_i^{(0)}$ denotes the initial value of x_i. A round r is *quiescent with regard to* process p_i, also noted *i-quiescent*, if p_i detects no additional failure during round r.

For termination, consider a run of the algorithm in which exactly f processes crash ($0 \leq f \leq t$). Every correct process p_i detects f crashes in this run, and so there is at least one round r_i with $1 \leq r_i \leq f+1$ which is i-quiescent, i.e., $Rec\_Failed_i^{(r_i)} = Failed_i^{(r_i)}$. Suppose p_i has not yet decided at the end of round $r_i - 1$; then we have $halt_i^{(r_i-1)} = \texttt{false}$. There are two cases to consider:

1. Process p_i is not ready at the end of round $r_i - 1$, that is $ready_i^{(r_i-1)} = \texttt{false}$. Since round r_i is i-quiescent, p_i necessarily becomes ready at round r_i, and so p_i decides at the end of round $r_i + 1$ if $r_i \leq t-1$ and at the end of round r_i if $r_i \geq t$.
2. Process p_i is ready at the end of round $r_i - 1$. In this case, the algorithm forces p_i to decide the current value of $W_i[i]$ at the end of round r_i.

In any case, p_i decides by the end of round $f+2$ if $0 \leq f < t-1$ and by the end of round $f+1$ if $t-1 \leq f \leq t$. This proves that the algorithm meets the lower bounds of Theorem 1, and subsequently the termination property.

For agreement, we first give a lemma which establishes a weak form of agreement among the processes that are ready. More precisely, the lemma states that two *ready* messages sent at the same round carry the same value.

[3] V may be an arbitrary non-empty set when examining termination and agreement (Theorem 3.1), but must be instantiated to $\{0, 1\}$ for the k-validity (Theorem 3.2).

Lemma 1. *If two messages* $(ready, v)$ *and* $(ready, v')$ *are sent at the same round* $r \geq 2$*, then* $v = v'$.

Proof: The proof is by induction on r. Let p_i and p_j be the senders of $(ready, v)$ and $(ready, v')$, respectively; from the algorithm, p_i and p_j have not yet halted but have both become ready at the end of round $r - 1$.

Basis: $r = 2$. Processes p_i and p_j can send a *ready* message at round 2 only if the first round is i and j-quiescent. In other words, both p_i and p_j receive the same vector W of n initial values at round 1, and so $W_i[i]^{(1)} = W_j[j]^{(1)} = \Phi(1, W)$. From the code of the algorithm, p_i and p_j send *ready* messages at round 2 which are tagged by $W_i[i]^{(1)}$ and $W_j[j]^{(1)}$, respectively. Therefore, we have $v = v'$.

Inductive step: Assume $r > 2$, and suppose the lemma holds at any round r' with $2 \leq r' < r$. There are three cases to consider:

1. Processes p_i and p_j both receive some *ready* messages at round $r - 1$. From the inductive hypothesis, all these *ready* messages are tagged by the same value ν. Upon receiving $(ready, \nu)$ at round $r-1$, p_i and p_j set $W_i[i] := \nu$ and $W_j[j] := \nu$, respectively. At round r, p_i and p_j both broadcast *ready* messages which carry the current value of $W_i[i]$ and $W_j[j]$, i.e., the value ν. Therefore, we have $v = \nu = v'$.
2. Exactly one of the two processes, say p_i, receives some *ready* messages at round $r - 1$; the same argument as before shows that each message is tagged with value ν. Let p be the sender of such a message; process p is obviously alive at the very beginning of round $r-2$. On the other hand, since p_j becomes ready at round $r-1$ without receiving a *ready* message, it follows that round $r - 1$ is j-quiescent. At round $r - 1$, process p_j thus detects no new failure and so receives a message from p. This message is equal to $(ready, \nu)$ since p sends the same message to all processes. This contradicts that only p_i receives a *ready* message at round $r - 1$, and so this case cannot occur.
3. Neither p_i nor p_j receives *ready* messages at round $r-1$. From the algorithm, it follows that round $r - 1$ is both i and j-quiescent. Then p_i and p_j receive the same information at this round, and so the W_i and W_j vectors are equal just after the receive phase of round $r-1$. Let W denote this vector. Since neither p_i nor p_j receive a *ready* message at round $r - 1$, both p_i and p_j set their own component to $\Phi(r-1, W)$, that is $W_i[i]^{(r)} = W_j[j]^{(r)} = \Phi(r-1, W)$. Then p_i and p_j tag the *ready* messages that they send at round r with the current values of $W_i[i]$ and $W_j[j]$, respectively. Therefore, we have $v = W_i[i]^{(r-1)}$, $v' = W_j[j]^{(r-1)}$, and so $v = v'$ as needed.

$\Box_{Lemma\ 1}$

Now we prove that the algorithm in Figure 1 satisfies agreement. Let r be the first round in which some process decides. Let p_i denote this process and v its decision value. Let p_j be a process that decides v' at round r'. Thus, we have $r' \geq r$. We consider two cases:

1. Process p_i is ready at the end of round $r - 1$, and so sends a *ready* message at round r. From the algorithm, this message is tagged by $W_i[i]^{(r-1)}$. Afterwards, p_i does not modify the value of $W_i[i]$, and so decides $v = W_i[i]^{(r-1)}$. The $(ready, v)$ message from p_i is received by p_j since p_j is still alive at the end of round r', and $r' \geq r$.
   (a) If p_j is also ready at the end of round $r - 1$, then it sends a *ready* message at round r. Therefore, $r = r'$ and the *ready* message from p_j is tagged by the current value of $W_j[j]^{(r-1)}$. Following the same argument as for p_i, process p_j decides value $v' = W_j[j]^{(r-1)}$. Lemma 1 shows that $W_i[i]^{(r-1)} = W_j[j]^{(r-1)}$, and so $v = v'$.

(b) Otherwise, p_j receives some *ready* messages at round r, and by Lemma 1 all these messages are tagged by v. From the algorithm, p_j must set $W_j[j]$ to v and becomes ready at the end of round r. Then p_j decides the current value of $W_j[j]$ at round r or $r+1$. This implies that $v' = v$, as needed.

2. Process p_i is not ready at the end of round $r-1$. This may happen only if $r \geq t$, i.e., $r = t$ or $r = t+1$. Moreover, we can assume that p_j is also not ready at the end of round $r-1$ (otherwise, we have $r' = r$ and then we go back to the first case by exchanging the roles of p_i and p_j). Therefore, none of the rounds $1, \cdots, r-1$ are i or j-quiescent. There are two sub-cases to consider:
   (a) Process p_i receives some *ready* messages at round r. Then all these messages are tagged by the decision value v of p_i. Let p_k be the sender of such a message.
      i. Assume process p_j also receives $(ready, v)$ from p_k at round r; then p_j becomes ready and sets $W_j[j]$ to v. Since $r \geq t$, process p_j decides at the end of round r and its decision value is the current value of $W_j[j]$, i.e., $W_j[j]^{(r)} = v$. This proves that $v' = v$.
      ii. Otherwise process p_j receives no message from p_k at round r. Since p_k sends a message to p_i at this round, it necessarily succeeded in sending a message to p_j at the previous round $r-1$. Therefore, round r is not j-quiescent, and so p_j does not decide at round r. Moreover, p_j has detected at least r crashes at the end of round r. Since $r \geq t$, no additional failure can occur later. This shows that process p_i (that is still alive at the end of round r) is actually correct, and that it succeeds in sending $(ready, v)$ to p_j at round $r+1$. It follows that p_j decides the value $v' = v$ at round $r+1$.
   (b) Process p_i receives no *ready* message at round r. From the algorithm, it follows that round r is i-quiescent.
      i. If process p_j receives a *ready* message at round r, then p_j becomes ready and since $r \geq t$, it decides at round r. We get $r' = r$, and by exchanging the roles of p_i and p_j we go back to the previous sub-case 2(a)ii.
      ii. Otherwise process p_j receives no *ready* message at round r. At this point there are two possible cases:
         A. Round r is also j-quiescent. According to the algorithm, p_j decides at round r, and so $r' = r$. The fact that round r is both i and j-quiescent ensures that p_i and p_j receive the same information at this round, and so vectors W_i and W_j are equal just after the receive phase of round r. Let W denote this vector. Since neither p_i nor p_j receive a *ready* message at round r, both p_i and p_j set their own component to $\Phi(r, W)$, that is $W_i[i]^{(r)} = W_j[j]^{(r)} = \Phi(r, W)$. Then p_i and p_j decide on $v = W_i[i]^{(r)}$ and $v' = W_j[j]^{(r)}$, respectively. This ensures that $v = v'$.
         B. Otherwise, p_j detects a new crash at round r. Therefore, p_j has detected at least r crashes at the end of round r. We then argue as in a previous case (2(a)ii). Since $r \geq t$, no additional failure can occur later. This shows that process p_i which is still alive at the end of round r, is actually correct, and that it succeeds in sending $(ready, v)$ to p_j at round $r+1$. It follows that p_j decides the value $v' = v$ at round $r+1$.

$\Box_{Theorem\ 2}$

We now address k-validity (and so, $V = \{0, 1\}$). For that, we constrain the set to which Φ belongs. More precisely, for each $k \in \{1, \cdots, n\}$ we restrict ourselves to functions Φ that satisfy the following condition $\mathcal{C}_k$:

(1) $\Phi(r,W) = 0$ if $r = 1$ and $|\{i : W[i] \in \{0,\perp\}\}| \geq k$;
(2) $\Phi(r,W) \in \{W[i] : W[i] \neq \perp\}$ otherwise, when moreover $W \neq (\perp,\cdots,\perp)$.

Note that for every $k \in \{1,\cdots,n\}$, there exist functions satisfying $\mathcal{C}_k$. Moreover, part (2) in $\mathcal{C}_k$ is natural since Φ is never applied to $(\perp,\cdots,\perp)$ during the algorithm (cf. Section 3).

We now prove that condition $\mathcal{C}_k$ for function Φ in Figure 1 shows an algorithm which solves the k-TAg problem.

Theorem 3. *For any function $\Phi : \{1,\cdots,t+1\} \times (V \cup \{\perp\})^n \rightarrow V$ satisfying $\mathcal{C}_k$, the algorithm in Figure 1 satisfies the k-validity property.*

Proof: First suppose that at least k processes start with the initial value 0. Let p_i be any process. Process p_i receives value 0 or no message at the first round from at least k processes. From the definition of $\Phi_k(1,-)$, it follows that $W_i[i]^{(1)} = 0$. By the algorithm, any component $W_j[j]$ remains equal to 0 in the subsequent rounds. So, 0 is the only possible decision value.

On the other hand, consider a run of the algorithm in which every process starts with 1, and with at most $k-1$ failures. In the first round, every process p_i receives at least $n-k+1$ messages with value 1, and so $W_i[i]^{(1)} = 1$ (if p_i is still alive at the end of round 1). A simple induction on r shows that for any r, we have $W_i[i]^{(r)} = 1$ because of the definition of $\Phi_k(r,-)$ for $r \geq 2$. Therefore, the only possible decision value in this run is 1.

$\Box_{Theorem\ 3}$

For $k = n$, the algorithm in Figure 1 thereby solves binary Consensus. Obviously, this algorithm can be generalized for an arbitrary value set V.

Part (2) of $\mathcal{C}_k$ allows flexibility in the choice of Φ. For instance, min, max, and $W[i]$ where $i = \min\{j : W[j] \neq \perp\}$ all yield correct solutions of the k-TAg problems. In the Non-Blocking Atomic Commitment case, this flexibility can be used to tune the frequency of commit decisions. Indeed, setting $\Phi(r,W) = \max_i W[i]$ in part (2) of $\mathcal{C}_k$ allows processes to commit more often: if all processes initially vote `yes`, then the decision is `commit` as soon as at least one correct process has received `yes` from all processes in the first round no matter whether failures occur. This is clearly a desirable feature of any database transaction system.

Conclusion

In this paper, we introduced a new class of distributed agreement problems, ranging from Consensus to Non-Blocking Atomic Commitment, by varying the validity condition in the specification. We also provided an early deciding algorithm to solve each problem of this class in the synchronous model with crash failures.

Lower bounds on time-complexity have already been established for early deciding algorithms for both Consensus and Non-Blocking Atomic Commitment, and can be easily extended to any k-TAg problem. In this paper, we proved that our algorithm meets these lower bounds, so showing that these lower bounds are tight.

These results show that Consensus and Non-Blocking Atomic Commitment, two incomparable agreement problems in asynchronous systems, can be solved by the same optimal algorithm in synchronous systems. Consequently, the common belief that Non-Blocking Atomic Commitment requires a three-phase commit protocol "*à la Skeen*" is false.

Acknowledgments

We are grateful to Leslie Lamport and James Leifer for their very constructive and helpful comments.

References

1. B. Charron-Bost and A. Schiper. Uniform consensus is harder than consensus. Technical Report DSC/2000/028, Département Systèmes de Communication, EPFL, May 2000. To appear in Journal of Algorithms.
2. B. Charron-Bost and S. Toueg. Comparing the atomic commitment and consensus problems. In preparation, 2001.
3. C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.
4. C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, October 1990.
5. M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
6. I. Keidar and S. Rajsbaum. A simple proof of the uniform consensus synchronous lower bound. *Information Processing Letters*, 85(1):47–52, January 2003.
7. L. Lamport. The weak byzantine generals problem. *Journal of the ACM*, 30(3):668–676, July 1983.
8. L. Lamport. Lower bounds on consensus. Unpublished note, March 2000.
9. L. Lamport and M. Fischer. Byzantine generals and transaction commit protocols. Technical Report 62, SRI International, April 1982.
10. N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.
11. M. J. Merritt. Unpublished Notes, 1985.
12. D. Skeen. Nonblocking commit protocols. In *Proceedings of the ACM SIGMOD Conf. on Management of Data*, pages 133–147. ACM, June 1982.