SIAM J. COMPUT.
Vol. 40, No. 1, pp. 63–78

THE COMPLEXITY OF EARLY DECIDING SET AGREEMENT*

ELI GAFNI†, RACHID GUERRAOUI‡, AND BASTIAN POCHON‡

Abstract. In the k-set agreement problem, each processor starts with a private input value and eventually decides on an output value. At most k distinct output values may be chosen, and every processor's output value must be one of the proposed values. We consider a synchronous message passing system, and we prove a tight bound of $\lfloor f/k \rfloor + 2$ rounds of communication for all processors to decide in every run in which at most f processors fail. The lower bound proof proceeds through a simulation of a synchronous solution to k-set agreement in message passing, in an asynchronous shared memory system in which $k-1$ processors may fail, and which was proven to be impossible using topological approaches. In contrast to past complexity results on set agreement, our lower bound proof is purely algorithmic. It does not use any direct topological argument but uses instead the impossibility of asynchronous set agreement to encapsulate the needed topology. We thus derive an adaptive complexity lower bound for a message passing system from a static impossibility in a shared memory system.

Key words. distributed algorithm, set agreement, lower bound, message passing system, shared memory system, simulation

AMS subject classification. 68

DOI. 10.1137/050640746

1. Introduction.

Results about the complexity of set agreement are intriguing, as they present an intrinsic trade-off between the degree of coordination that these processors can reach and the number of failures that are tolerated [7]. The complexity of early deciding [9] set agreement is even more intriguing as it brings to the picture the number of failures that *actually* occur in a given computation.

1.1. Set agreement.

Set agreement is a natural generalization of the widely studied consensus problem [11]. In set agreement, just like in consensus, each processor is supposed to propose a value and eventually decide on some output that was initially proposed, such that every correct processor eventually decides. Processors are restricted not to decide on more than k distinct outputs. We talk about k-set agreement, and consensus is the special case where $k = 1$.

Set agreement was introduced in [6], where it was conjectured that, in an *asynchronous* system, the problem has a solution if and only if strictly less than k processors may crash. (An asynchronous model of distributed computation is characterized by the following two properties: (1) processors execute the algorithm assigned to them unless they crash, in which case they stop all their activities and they are said to be faulty (not correct); (2) processors make no assumption on their relative speed and message communication delay.) This conjecture has sparked a fruitful line of research [3, 14, 17].

*Received by the editors September 20, 2005; accepted for publication (in revised form) October 18, 2010; published electronically January 11, 2011. This paper is a revised version of a paper by the same authors entitled "From a static impossibility to an adaptive lower bound: The complexity of early deciding set agreement," which appeared in the *Proceedings of the 37th Annual ACM Symposium on Theory of Computing* (STOC'05).

http://www.siam.org/journals/sicomp/40-1/64074.html

†Department of Computer Science, UCLA, Los Angeles, CA 90024 (eli@cs.ucla.edu).

‡School of Computer and Communication Sciences, EPFL, 1015 Lausanne, Switzerland (rachid.guerraoui@epfl.ch, bastian.pochon@gmail.com).

1.2. The complexity of set agreement. After proving the impossibility of k-set agreement with k crashes in an asynchronous system, researchers turned to the complexity of the problem in the *synchronous* system [7, 12, 13]. (In a synchronous model of distributed computation, processors execute in a lockstep manner, moving incrementally from one round to the next, and exchanging messages in every round; if a processor p does not receive a message from a processor q in a round r, then no processor ever receives any message from q in any subsequent round $r' > r$.) Here, topological arguments were used to prove a fundamental complexity result.

In short, the result states that any synchronous k-set agreement algorithm that tolerates t failures (where $t < N$ and N is the total number of processors in the system) has at least one run where at least one correct processor does not reach a decision before round $\lfloor t/k \rfloor + 1$.

This lower bound does not, however, say much about the existence of algorithms that would expedite a decision in runs where f ($f \leq t$) failures actually occur. In particular, one would expect that, in runs where few failures occur, a decision can be reached *earlier* than in those with more failures. Algorithms that have such an adaptive flavor are called *early deciding*: their efficiency depends on the effective number of failures that occur in a given computation, rather than (only) on the (total) number of failures that can be tolerated [9].

In practice, failures rarely happen, and it makes sense to devise algorithms that decide earlier when fewer failures occur. For consensus, a significant efficiency improvement has been established when considering the effective number of failures [5, 10, 15]. In particular, it was shown that there is an algorithm where all correct processors decide by round $\min(f+2, t+1)$ in any run with at most f failures for any integer f [5, 15]. It was also shown that, for any integer $f \leq t$ and for any consensus algorithm, not every processor may decide before $\min(f+2, t+1)$ rounds in any run with at most f failures.

No (tight) bound for early deciding set agreement has been established so far. This might not be surprising given the involvement of the lower bound proofs of less general, nonearly deciding set agreement algorithms. In particular, when facing the question of a lower-bound to early deciding set agreement, it is not clear how to extend the topological arguments in [14]. The "protocol complex"—the topological entity used in that paper—applies to the "end" of the protocol. With early deciding, we deal with some processors outputting a decision in a round and some not: so there is no well-defined "end."

More generally, the topological method typically characterizes the structure of views of processors at the end of the computation and has not been adapted yet to deal with evolving computation. In a sense, early decision argumentation is evocative of the analysis that is called for when arguing one-shot versus long-lived object implementations [2]. The issue there is whether a processor can obtain an output in the face of continual arrival and departure of other processors. To date, the topological method has not helped in resolving this matter. The early decision question seems to fall into this category of evolving dynamic computations.

Also, here we consider set agreement; yet, even for consensus, the early deciding synchronous lower bound is involved. The consensus bound has been argued recently [15] through similarity between computations, rather than by using the more modern methods developed with the emergence of the topological techniques [3, 4, 7, 14].

1.3. Contribution. In this paper, we propose dealing with the dynamic situation of early decision in an iterative manner, rather than through a head-on attack. We do not apply topology directly. Rather, we reason about the dynamics of the synchronous computation through reduction. Unlike [12], where the simulation proceeds forward without "looking back," we propose a simulation technique using the BG-simulation [3, 4]. This allows simulators to go back and look at the transcript of the simulation and by that allows us to argue about the *dynamics* of the computation rather than just its *end*. Interestingly, even if the transcript is explored in the more abstract shared memory model, the lower bound we derive is for a message passing model.

More specifically, this paper first considers an underlying synchronous message passing system and exhibits a set-agreement algorithm where every correct processor decides within $\lfloor f/k \rfloor + 2$ rounds in every run in which at most f processors fail. Then we prove our main lower-bound result showing that the algorithm is optimal. The lower bound is expressed in general terms without any assumption on the total number of processors. Its proof proceeds through a simulation of a synchronous solution to k-set agreement in message passing, in an underlying asynchronous shared memory system in which $k-1$ processors may fail. The lower bound then derives from the impossibility of k-set agreement in the latter model, which has already been proved, e.g., using topological approaches.

Our result supports the tradition in computer science that once a few cornerstone impossibility or complexity results have been proved using direct arguments, from there one should use reductions rather than argue anew. In our distributed computing context, this translates into a minimal usage of topological techniques, in the same vein that one proves NP-completeness by reduction rather than rehashing Cook's proof of the SAT NP-completeness [8].

1.4. Roadmap. The rest of the paper is organized as follows. Section 2 gives some preliminaries about the distributed computing models that are needed to state and prove our results. Section 3 presents an algorithm for early deciding set agreement, giving the upper bound. Section 4 states and proves our lower-bound result, showing the bound to be tight. Section 5 compares our result with previous results on consensus. Section 6 concludes the paper with some final remarks.

2. Models and problems. In the following, we present the main elements of the synchronous message passing model, in which we state the lower bound and design our optimal early deciding set agreement algorithm. Then we present the asynchronous shared memory model, which we use in our lower-bound proof. (Remember that we reduce the lower bound on the synchronous complexity of early deciding set agreement in message passing into its asynchronous impossibility in shared memory.) We finally also briefly recall the set agreement problem.

2.1. The synchronous message passing model. We consider a set of processors $\Pi = \{p_0, p_1, \ldots\}$. Processors communicate by message passing. Communication channels are reliable. Processors execute in a synchronous, round-based model [16].

A run is a sequence of rounds. Every round is composed of three phases. In the first phase, every processor broadcasts a message to all the other processors. In the second phase, every processor receives all the messages sent to it during the round. In the third phase, every processor performs a local computation before starting the next round. Processors may fail by crashing.

A processor that crashes does not execute any step thereafter and is said to be

faulty. Processors that do not crash are said to be *correct*. When processor p_i crashes in round r, any subset of the messages that p_i sends in round r (possibly the empty set) might not be received by the end of round r. A message broadcast in round r by a processor that does not crash in round r is received, at the end of round r, by every processor that reaches the end of round r.

We say that a processor p_i *sees* f crashes at the end of any round r if p_i receives messages from all processors but f of them. We consider that at most $t < N$ processors may fail, i.e., crash, in any run. The *state* of a processor p_i, at the end of round r, consists of the content of its local memory, including the messages received in each round $r' \leq r$ as well as the local variables of p_i.

For simplicity, we sometimes say that an algorithm is synchronous (resp., asynchronous) to mean that the algorithm assumes a synchronous message passing (resp., asynchronous shared memory) model.

2.2. The asynchronous shared memory model. We prove our lower bound result by reducing computations in the synchronous message passing model, recalled above, to computations in the more abstract asynchronous shared memory model, which we recall here.

For clarity, processors are called *simulators* in the asynchronous shared memory model. Precisely, we consider a set of $k+1$ simulators $\{sim_0, \ldots, sim_k\}$.

Simulators communicate through asynchronous shared memory. In the asynchronous shared memory model, there exists no bound on the processor relative run speed. Shared memory is organized into cells (sometimes called registers), where each memory cell may contain an infinite number of bits.

Cells of the shared memory support three operations: the write(v) operation atomically writes value v into the cell; the read() operation atomically returns the content of the cell; and the snapshot() operation returns an atomic view of all the cells (i.e., the snapshot which can be implemented from read and write operations in asynchronous shared memory returns values of cell read simultaneously at some instance between the invocation and the return of the snapshot operation [1]). Any cell may be written by a single simulator and read by all of them.

For the sake of simplifying the presentation, in what follows we adopt the convention that, after executing an operation snapshot(*args*), the variables *args* are accessible by the simulator in its local memory, and the content of the *args* variables are the same as that at the time of the snapshot() operation.

Without loss of generality, we consider that the simulators execute full-information protocols in shared memory [14]. In a full-information protocol, any simulator sim_i writes its entire state into a memory cell whenever sim_i writes anything into this cell. Any simulator that later reads the cell reads the entire history of the states of simulator sim_i.

2.3. The *k*-set agreement problem. Each processor proposes a value v from a set of inputs V, and is supposed to eventually decide on an output v' of V, such that the following hold:

Validity. Every output is a proposed input value.

k-set agreement. There are at most k distinct outputs.

Termination. Every correct processor eventually decides on an output.

Solving k-set agreement in a wait-free manner means that every correct processor eventually decides (no matter how many processors fail). Wait-free k-set agreement is proved impossible in an asynchronous shared memory model of $k+1$ processors [3, 14, 17].

```
At processor p_i:
 1: halt := ∅ ; deciding := false
 2: S^r := ∅ ∀r

 3: procedure propose(v_i)
 4:    est_i := v_i
 5:    for r from 1 to ∞ do
 6:       if deciding then
 7:          send (r, DEC, est_i) to all
 8:       else
 9:          send (r, EST, est_i) to all
10:       if deciding then
11:          decide(est_i) ; return
12:       else if received any (r, DEC, est_j) then
13:          est_i := est_j ; deciding := true
14:       else
15:          S^r := {(est_j, j) | (r, EST, est_j) is received in round r from p_j}
16:          halt := Π \ ∪_{(est_j,j)∈S^r} {j}
17:          est_i := min{est_j | (est_j, j) ∈ S^r}
18:          if |halt| < rk then
19:             deciding := true
```

FIG. 3.1. *An early deciding k-set agreement algorithm (algorithm for processor p_i).*

3. An upper bound for early deciding set agreement. We give in Figure 3.1 an algorithm that solves k-set agreement in the synchronous message passing system $\Pi = \{p_0, p_1, \ldots\}$. In any run where there are at most f crashes, every correct processor decides within $\lfloor f/k \rfloor + 2$ rounds. Moreover, if there are eventually at most $k-1$ crashes per round, every correct processor eventually decides.

The algorithm is pretty simple. It helps us understand the notion of early decision and hopefully motivates the ideas underlying the lower-bound proof, which is the main technical contribution of this paper.

3.1. Overview of the algorithm. The idea underlying the algorithm is that each processor maintains its current estimate of the decision value, and in each round, processors exchange their estimate values. Moreover, each processor p_i maintains a set (denoted $halt_i$ in Figure 3.1) with the identity of the processors from which p_i does not receive a message.

If, in a given round r, less than k new processors crash, then, for any processor p_i, there are at most $k-1$ values among those remaining in the system that p_i has never received in any message (i.e., p_i is unaware of at most $k-1$ values of those remaining in the system).

Any processor p_i that misses at most $k-1$ values from those in the system in any round r has a decision value as its estimate at the end of round r. If p_i succeeds in sending its estimate value in the next round $r+1$ to every other processor without crashing, p_i may safely decide on its estimate value at the end of round $r+1$.

In the algorithm, processors decide only when there are at most $k-1$ crashes in a round r for any $r > 0$. (The decision effectively occurs at the end of round $r+1$.) However, processors do not decide if there are k crashes in the round.

3.2. Correctness of the algorithm. In the following, we denote the local copy of a variable *var* at processor p_i by var_i, and the value of var_i at the end of round r by var_i^r. We denote by $crashed^r$ the set of processors that crash *before* completing round r, and by $ests^r$ the set of estimate values of every processor at the end of round r. By definition, round 0 ends when the algorithm starts.

No processor decides by round 0. We first prove three general claims about the algorithm (of Figure 3.1).

CLAIM 3.1. $ests^r \subseteq ests^{r-1}$.

Proof. The proof is straightforward: for any processor p_i, $est_i^r \in ests^{r-1}$. ∎

CLAIM 3.2. *If at the end of round* $0 \leq r \leq \lfloor t/k \rfloor$ *no processor has decided, and at most* l *processors crash in round* $r+1$*, then* $|ests^{r+1}| \leq l+1$.

Proof. Assume the conditions of the claim, and assume by contradiction that $|ests^{r+1}| \geq l+2$. By assumption, $l+2$ processors have distinct estimate values at the end of round $r+1$. Denote by $q_0, \ldots, q_{l+1}$ these processors, such that $est_{q_i}^{r+1} \leq est_{q_{i+1}}^{r+1}$ for $0 \leq i \leq l+1$. Processors $q_0, \ldots, q_l$ do not send $est_{q_0}^{r+1}, \ldots, est_{q_l}^{r+1}$ in round $r+1$; otherwise, q_{l+1} receives one of the smallest $l+1$ estimate values in round $r+1$. Thus, $l+1$ processors send values corresponding to $est_{q_0}^{r+1}, \ldots, est_{q_l}^{r+1}$ in round $r+1$ and which crash in round $r+1$; otherwise, q_{l+1} receives one of the smallest $l+1$ estimate values in round $r+1$. This contradicts our assumption that at most l processors crash in round $r+1$. ∎

CLAIM 3.3. *If, at the end of round* $r \geq 1$*, no processor has decided, and* $|ests^r| \geq k+1$*, then* $|crashed^r| \geq rk$.

Proof. We prove the claim by induction. For the base case $r = 1$, assume that the conditions of the claim hold. That is, at the end of round 1, $k+1$ distinct processors $q_0, \ldots, q_k$ have distinct estimate values. By Claim 3.2, $|crashed^1| \geq k$. Assume the claim for round $r-1$, and assume the conditions of the claim hold at round r. We prove the claim for round r. By assumption, $k+1$ processors $q_0, \ldots, q_k$, at the end of round r, have $k+1$ distinct estimates. By Claim 3.1, $k+1$ processors necessarily reach the end of round $r-1$ with $k+1$ distinct estimates. Thus Claim 3.3 holds at round $r-1$ (induction hypothesis), and thus, $|crashed^{r-1}| \geq (r-1)k$. By Claim 3.2, at least k processors crash in round r. Thus $|crashed^r| \geq k + |crashed^{r-1}| \geq rk$. ∎

We are now able to prove the correctness of the algorithm.

THEOREM 3.4. *The algorithm in Figure* 3.1 *solves* k*-set agreement.*

Proof. Validity is obvious, as any processor p_i assigns its proposed value as its estimate, thereafter exchanges estimate values with other processors, and decides on its estimate value.

For proving that the algorithm ensures k-set agreement, consider the lowest round r in which some processor decides. If such a round r does not exist, then no processor ever decides, and k-set agreement is trivially satisfied. Hence we assume hereafter that such a round r exists. Let p_i be one of the processors that decide in round r. Processor p_i decides at line 11, after executing line 19 in round $r-1$, where *deciding* is set to true at processor p_i. In round $r-1$, p_i executes line 19 only if $|crashed|^{r-1} < rk$ holds at line 18. Thus, from Claim 3.3, there are at most k distinct estimate values at the end of round $r-1$ in the system, which ensures k-set agreement.

For proving that, if there are eventually at most $k-1$ crashes per round, then every correct processor eventually decides, observe that, if in any round r at most $k-1$ processors crash, then $|halt_r| < rk$ at the end of round r. Hence, every correct processor executes line 19, sends its estimate as a decision in round $r+1$, and decides on its estimate value at the end of round $r+1$ at line 11 in the algorithm. ∎

THEOREM 3.5. *In any run with at most f failures, any correct processor decides by round $\lfloor f/k \rfloor + 2$ with the algorithm in Figure* 3.1.

Proof. Assume a run with at most f failures. By way of contradiction, assume that there exists a processor p_i for which $|halt_i^r| \geq rk$ for $r = \lfloor f/k \rfloor + 1$. (Otherwise, if $|halt_i^r| < rk$, then p_i decides at line 11 in the next round.) Processor p_i does not decide in round r; in particular, p_i does not receive any DEC message in round r. We have $|halt_i^r| \geq rk = (\lfloor f/k \rfloor + 1)k = \lfloor f/k \rfloor k + k > f$, which is a contradiction. □

4. The lower bound. Our lower-bound proof proceeds through a simulation of a synchronous solution to k-set agreement in message passing, in an underlying asynchronous shared memory system in which $k-1$ processors may fail. The lower bound then derives from the impossibility of k-set agreement in the latter model.

The simulation relies on a reduction technique called the BG-agreement protocol [3, 4]. For completeness and self-containment of our lower bound proof we briefly review this protocol here before giving our main lower-bound result. (A formal treatment of the protocol is given in [4].)

4.1. The BG-agreement protocol. A BG-agreement protocol is a distributed algorithm involving a set of processors (called simulators here) that seek to reach agreement in the asynchronous shared memory model. The protocol has exactly one wait statement—the last one.

The BG-agreement protocol consists in deciding one of the values proposed by the simulators. The simulator whose value is decided is called the winner of the protocol. The protocol is guaranteed to decide a value when all participating simulators arrive at the wait statement. These participating simulators are not known in advance. While waiting for other simulators to reach the wait statement, the outcome of the protocol may not be known and, in our terminology, we say that the BG-agreement is not *resolved*.

The crucial observation here is that, if the BG-agreement is not resolved, then one of the participating simulators is in the middle of the algorithm rather than at the wait statement (we say that this simulator is *blocking* the BG-agreement). Interestingly, if simulators that are waiting time-share and execute other protocols, and the BG-agreement is not resolved, we can conclude that at least one simulator does not participate in other protocols.

An implementation of the BG-agreement protocol is illustrated in Figure 4.1. Variables v_i, x_i, and S_i (for any $0 \leq i \leq n$) are in shared memory, are written by simulator sim_i, and are read by all. The $*$ in front of the parameter *result* indicates an output parameter. The wait statement spans over lines 11 to 13.

A simulator proposes a value v to a BG-agreement instance by invoking BGpropose(v, *result*) and expects the result of the agreement to be stored in local variable *result*.

The intuitive idea underlying how the BG-agreement protocol works in Figure 4.1 is as follows: a simulator (i) writes its proposed value and its identifier in shared memory (we say that the simulator "registers"), (ii) takes a snapshot of the registered simulators, and (iii) writes its snapshot into shared memory. The simulator then continuously takes snapshots of the shared memory until all the registered simulators have written their snapshots into shared memory. The simulator then returns the value of the simulator with the smallest identifier in the smallest set corresponding to the snapshot of a simulator. This simulator with the smallest identifier is the winner.

Notice that, in the BG-agreement protocol, a simulator, after taking a snapshot, has a set of candidate winners—those that appear in its snapshot. No simulator

```
1: In shared memory:
2:       v_i ∈ V, init ⊥
3:       x_i ∈ {true, false}, init false
4:       S_i ⊆ {0, ..., n}, init ∅

5: procedure BGpropose(v, *result)
6:    v_i := v
7:    x_i := true
8:    snapshot(x_1, ..., x_n)
9:    S_i := { j | x_j = true, 0 ≤ j ≤ n }
10:   do
11:   { The do loop is the wait statement }
12:      snapshot(S_0, ..., S_n)
13:   until ∀j ∈ S_i : S_j ≠ ∅
14:   winner := min(S_j), where j ∈ S_i and ∀k ∈ S_i : |S_k| ≥ |S_j|
15:   *result := v_winner
```

FIG. 4.1. *BG-agreement protocol (algorithm for simulator sim_i).*

registering later may win the agreement, and, more generally, no simulator registering after any other processor arrived at the wait statement may win the agreement. Thus, if a simulator, after arriving at the wait statement, observes that all current proposals are the same, this simulator may determine the resolution of the agreement. In this sense, a BG-agreement instance is an "open" box. Any simulator may access the shared memory used in a particular BG-agreement instance without invoking BGpropose, e.g., to read all the proposals to this instance and determine the winner of this instance.

4.2. Main theorem. We now give the main theorem behind our lower-bound result. The theorem is expressed in general terms, without any mention of the total number of processors.

THEOREM 4.1. *For any integer $f \geq 0$, no synchronous algorithm $C(k, f)$ solves k-set agreement under the following conditions:*

1. *In runs in which eventually no more than $k-1$ processors crash in each round, eventually every correct processor decides.*
2. *A processor that sees f failures (for some fixed f) decides within $\lfloor f/k \rfloor + 1$ rounds.*

The first condition can be viewed as a nontriviality property. Without it, Theorem 4.1 would be trivially wrong: for any integer $f \geq 0$, it is possible to design an algorithm where all processors that decide do so by round $\lfloor f/k \rfloor + 1$, in any run with at most f failures, if eventually there will be no failure.

The proof is by contradiction, and the main idea is to reduce the problem of solving wait-free k-set agreement with an asynchronous shared memory algorithm to a synchronous message passing algorithm $C(k, f)$ solving k-set agreement and satisfying the two conditions of Theorem 4.1. The impossibility of the former problem [3, 14, 17] implies the impossibility of the latter.

In short, the reduction consists in simulating, with algorithm $C(k, f)$, a run of an asynchronous shared memory algorithm that wait-free solves k-set agreement among $k + 1$ processors (simulators).

BG-agreement in $R_{r,1}$	
Purpose	agree upon the state of a processor p_j at the beginning of round $r+1$ (i.e., whether p_j crashes in round r and, if not, which messages p_j receives in round r)
Input values	"failed," "p_j receives messages from all processors in a set *correct* $\subseteq 2^{\Pi}$"

BG-agreement in $R_{r,2}$	
Purpose	agree upon a correct processor at the beginning of round $r+1$
Input values	"no processor," "kill $p_l \in \Pi$"

FIG. 4.2. *Series of BG-agreements in $R_{r,1}$ and $R_{r,2}$.*

4.3. Proof overview. We first give an intuitive idea of the simulation underlying the lower bound proof.

In the simulation of each synchronous round of algorithm $C(k, f)$, the $k+1$ simulators use a series of BG-agreement instances (Figures 4.3 to 4.7) to decide which messages any processor p_j received and which messages p_j did not receive; this determines the new state of p_j.

When a simulator sim_i decides, in any of the BG-agreements, that messages of p_j were not received, sim_i somehow *fails* processor p_j (we also say that p_j was chosen to be failed); this means that sim_i is simulating a run of $C(k, f)$ where processor p_j crashes.

The exact simulation performed by simulator sim_i depends on the execution of the particular BG-agreement, according to Figure 4.2. Any simulator sim_i that blocks a BG-agreement does not let the other simulators involved in the same BG-agreement decide upon the state of processor p_j; as a simulator may block at most one BG-agreement, in each round at most k BG-agreements may be unresolved.

In the simulation, this is translated into at most k new failures per round of the synchronous run. If a BG-agreement in the "far past" is not resolved, then a simulator is blocked in this BG-agreement, which means that the simulation proceeds from some round on with less than $k+1$ simulators and therefore generates less than k failures per round.

This, according to condition 1 of Theorem 4.1, forces the processors to decide and allows the simulators to read any processor decision and then decide on the same value.

On the other hand, if no simulator is blocked in any past BG-agreement, then a correct processor eventually decides according to condition 2 of Theorem 4.1. The simulators identify a processor that is correct and which decides according to condition 2. The simulators may read the decision of this processor and decide on the same value. The processor that decides does not interfere with the simulation after it decides. This is because the simulation ensures that this processor fails immediately after deciding.

4.4. Lower-bound proof. Assume by contradiction that algorithm $C(k, f)$, satisfying the two conditions of Theorem 4.1, exists. We show how $k+1$ simulators $sim_0, \ldots, sim_k$ solve k-set agreement asynchronously in a wait-free manner (i.e., while tolerating k simulator crashes) in shared memory, using C. This has been proved impossible [3, 14, 17].

```
1: In shared memory:
2:     state_{r,j}, init ⊥
3:     FinalFaulty_{r,j}, init ∅, r ≥ 1, 0 ≤ j ≤ n

4: procedure Simulate(C, f)
5:   r := 0, Correct := Π
6:   {
7:   {Execute two coroutines in parallel}
8:   ResolveInputs()
9:   {Coroutine 1: the simulation}
10:  for r := 1 to ∞ do
11:    r := +1
12:    Execute R_{r,1}
13:    Execute R_{r,2}
14:    SimulateRound(C, r)
15:  } || {
16:  {Coroutine 2: finding a decision}
17:  for scan := 1 to r do
18:    if ∃p_j ∈ Π : state_{scan,j} = "failed" then
19:      Correct := Correct − { p_j }
20:    if ∃p_j ∈ Π : state_{scan,j} = "decided v" then
21:      decide v
22:    else if ∃p_l ∈ Π : state_{scan,j} = "killed" then
23:      add or subtract messages to p_l from faulty processors to have exactly
24:      f failures
25:      resimulate C(k, f) with the new messages to p_l; p_l decides on v
26:      decide v
27:    else if |Correct| ≤ N − f then
28:      select the faulty processor p_l from which all correct
29:              processors receive a message in round scan
30:      add or subtract messages to p_l from faulty processors to have exactly
31:      f failures
32:      resimulate C(k, f) with the new messages to p_l; p_l decides on v
33:      decide v
34:  }
```

FIG. 4.3. *Simulation of algorithm C (algorithm for simulator sim_i).*

```
1: procedure ResolveInputs()
2:   for each p_j ∈ Π do
3:     BGpropose_{j,0}(i, state_{1,j})
```

FIG. 4.4. *Resolving inputs of algorithm C (algorithm for simulator sim_i).*

The proof is divided into three parts.

1. We first inductively show how a synchronous round $r > 1$ of algorithm $C(k, f)$ can be simulated in the asynchronous shared memory model, assuming round $r - 1$ was simulated.

```
1: procedure Execute R_{r,1}
2:    snapshot(state_{r,0}, ..., state_{r,n})
3:    F_{r,i} := { p_j | state_{r,j} ∈ {⊥, "failed", "killed"}} ∪ Suspected_{r,i}
4:    for each p_j ∈ F_{r,i} do
5:       BGpropose_{j,r,1}("failed", state_{r+1,j})
6:    snapshot(state_{r+1,0}, ..., state_{r+1,n})
7:    FinalFaulty_{r+1,i} := { p_j | state_{r+1,j} = "failed" or
8:          BGpropose_{j,r,1} has only "failed" proposals}
9:    for each p_j ∈ Correct_{r+1,i} := Π\FinalFaulty_{r+1,i} do
10:      BGpropose_{j,r,2}("p_j receives messages from all processors in Correct_{r+1,i}",
11:                        state_{r+1,j})
```

FIG. 4.5. *First asynchronous phase $R_{r,1}$ (algorithm for simulator sim_i).*

```
1: procedure Execute R_{r,2}
2:    snapshot(state_{r+1,0}, ..., state_{r+1,n})
3:    snapshot(FinalFaulty_{r+1,0}, ..., FinalFaulty_{r+1,n})
4:    if (i) p_l = ⊥ and
         (ii) ∃sim_q : |FinalFaulty_{r+1,q}| ≥ f and
         (iii) ∄(p_j ∈ Π, r' ≥ 1) : state_{r',j} = "killed" and
         (iv) ∄(p_j ∈ Π, r' ≥ 1) : BGpropose_{r'} has only "kill p_j" proposals then
5:       processorToKill := min_j{ p_j | state_{r+1,j} ∉ {⊥, "failed", "killed"}}
6:       BGpropose_r("kill processorToKill", p_l)
7:    else
8:       BGpropose_r("no processor", p_l)
9:    if p_l ∉ {⊥, "no processor"} then state_{r+1,l} := "killed"
10:   snapshot(state_{r+1,0}, ..., state_{r+1,n})
11:   for all proposed p_j ≠ p_l in BGpropose_r do
12:      Suspected_{r+1,i} := Suspected_{r+1,i} ∪ { p_j }
```

FIG. 4.6. *Second asynchronous phase $R_{r,2}$ (algorithm for simulator sim_i).*

2. We then exploit the two conditions of Theorem 4.1 so that each simulator can reach a decision with the simulation of $C(k, f)$ presented in the first part.
3. We finally show how to initiate the simulation by instantiating the first part with $r = 1$.

Proof. Part 1: *Simulating synchrony with asynchrony.* The simulators execute two asynchronous phases, $R_{r,1}$ and $R_{r,2}$, for every synchronous round r of algorithm $C(k, f)$. In the first asynchronous phase, $R_{r,1}$, simulating round r of $C(k, f)$, the simulators tentatively agree on the state of each processor at the end of round r or, equivalently, at the beginning of round $r + 1$ (i.e., on what messages are received by each of the processors in synchronous round r, if any). In the second asynchronous phase, $R_{r,2}$, simulating round r of $C(k, f)$, the simulators tentatively agree on a correct processor and simulate the failure of this processor at the beginning of round $r + 1$. Asynchronous phases $R_{r,1}$ and $R_{r,2}$ are executed using several BG-agreement instances, as depicted in Figures 4.3 to 4.7.

CLAIM 4.2. *For any integer $r > 1$, the algorithm in Figures* 4.3 *to* 4.7 *simulates a round r of synchronous algorithm $C(k, f)$, assuming it simulated round $r - 1$.*

```
1: procedure SimulateRound(C, r)
2:    execute round r of C using state_{r,0}, ..., state_{r,n}:
3:    • if a processor p_j decides on a value v, then state_{r+1,j} := "decided v"
4:    • otherwise generate the content of the messages to be sent in round r + 1
```

FIG. 4.7. *Simulating code C (algorithm for simulator sim_i).*

Proof. In the first asynchronous phase, $R_{r,1}$, every simulator sim_i takes a first snapshot and gathers in a set $F_{r,i}$ the processors (a) for which the state at the beginning of round r is not determined, or (b) for which the state at the beginning of round r is determined and indicates that the processor is faulty. Simulator sim_i proposes, in a first series of BG-agreements, in order to determine the state of each of the processors in $F_{r,i}$ at the beginning of round $r+1$, failing each processor in $F_{r,i}$. Simulator sim_i, after finishing all these BG-agreements (many of which are possibly unresolved), then takes a second snapshot and gathers in a set $FinalFaulty_{r+1,i} \subseteq F_{r,i}$ the processors in $F_{r,i}$ which are faulty at the beginning of round $r+1$. (These are the processors decided to fail by resolved BG-agreements, plus the processors in $F_{r,i}$ for which all proposals are to fail them in the corresponding BG-agreement.) For any processor p_j in the complement set $Correct_{r+1,i} = \Pi \backslash FinalFaulty_{r+1,i}$, sim_i obtains the state of p_j at the beginning of round r, from either the first or the second snapshot in $R_{r,1}$. Simulator sim_i writes $FinalFaulty_{r+1,i}$ in shared memory and then proposes, in a second series of BG-agreements in $R_{r,1}$, in order to determine the state of each of the processors in $Correct_{r+1,i}$, at the beginning of round $r+1$, that each processor in $Correct_{r+1,i}$ receives a message from every other processor in $Correct_{r+1,i}$. (Note that it is possible that a processor that belongs to $F_{r,i}$ for simulator s_i also belongs to $Correct_{r+1,i}$ thereafter in the case that another simulator proposes not failing this processor. In this case, simulator s_i does not use a second BG-agreement for determining processor p_i's state.)

Simulator sim_i now moves to the second asynchronous phase $R_{r,2}$; sim_i first takes a snapshot to observe the state of the processors at the beginning of round $r+1$. Then sim_i proposes a correct processor, using a single BG-agreement, whose purpose is to agree upon a correct processor. sim_i chooses a correct processor to propose as follows:

1. there is a simulator sim_q in the snapshot taken by sim_i, such that sim_q sees f or more processor failures, and
2. sim_i has not yet chosen a correct processor, nor observed such a processor being chosen, nor guaranteed to be chosen,[1] in a previous asynchronous phase $R_{s,2}$, $s < r$.

Otherwise there is no such processor and sim_i proposes a special "no processor" value. The idea is to simulate the failure of the processor agreed upon at the beginning of round $r+1$.

Following the BG-agreement of $R_{r,2}$ (not necessarily resolved yet), simulator sim_i takes a snapshot of the proposals to the BG-agreement of $R_{r,2}$ and starts to simulate synchronous round $r+1$. For each correct processor that appears in the last snapshot taken, that is, a correct processor that may be chosen as the result of the

[1] For instance, because a BG-agreement, though not yet resolved, may guarantee that a processor will be chosen if all propositions are to fail the same processor and that no proposition is to fail no processor.

BG-agreement in $R_{r,2}$, its state at the beginning of synchronous round $r+1$ is not determined until the BG-agreement of $R_{r,2}$ is resolved (the processor is "suspected"). Consequently, in $R_{r+1,1}$, all the simulators propose to fail these processors at the beginning of synchronous round $r+2$, that is, in the first series of BG-agreements in $R_{r+1,1}$. □

Part 2: *Reaching a decision.* We make the following claim.

CLAIM 4.3. *Consider the algorithm made of rounds $C(k,f)$ simulated in Figures* 4.3 *to* 4.7*, and assume that in runs in which eventually no more than $k-1$ processors fail in each round, eventually every correct processor decides; and a processor that sees f failures (for some fixed f) decides within $\lfloor f/k \rfloor + 1$ rounds. Then every correct simulator decides.*

Proof. Throughout the simulation, simulator sim_i continuously reads the shared memory in order of increasing rounds starting at round 1 to determine the first processor p_l that has been agreed upon as the result of $R_{r,2}$ for some round r. Because all simulators have the same rule for determining this processor, they will all agree on the same p_l (if one exists). There are two cases however, in which there may never be such a processor:

1. the simulation goes almost lockstep and less than f processors fail in the simulation, or
2. the simulators cannot determine p_l because a past BG-agreement is not yet resolved.

In any of these cases, there will eventually be less than k faulty processors per round. Therefore, the synchronous simulated processors eventually have to decide, according to the algorithm $C(k,f)$.

Now, suppose that none of these cases happen, i.e., every BG-agreement is eventually resolved, but there are forever synchronous rounds with k failures in each round (i.e., the opposite of eventually strictly less than k failures per round). Thus, the number of faulty processors grows without bound as the simulation proceeds far enough. In this case, when reading the shared memory, the simulators will all determine a round m such that m is the first round in which f or more processors are faulty at the beginning of round m. Since, for each simulator sim_i, each processor in the set $FinalFaulty_{m-1,i}$ is faulty at the beginning of round m, it follows that in round $m-2$ or less, no correct processor was chosen to fail in $R_{m-2,2}$. To see why, suppose by contradiction that a correct processor was chosen to fail at round $m-2$ (i.e., in $R_{m-2,2}$). Then at least one simulator sim_q has $FinalFaulty_{m-2,q} \geq f$. Since these processors will be faulty at the beginning of $m-1$, and additionally one correct processor was chosen to fail, there are more than f failures at the beginning of round $m-1$, contradicting the assumption that m is the first such round.

Since, at the beginning of round m, there are f failures or more, and at the beginning of round $m-1$, there are at most $f-1$ failures, there must be, at the beginning of round m, a processor p_j that fails, and all correct processors at the beginning of round m receive a message from p_j in round $m-1$. There are two cases:

1. A correct processor p_l is chosen by the BG-agreement in $R_{m-1,2}$.
2. No correct processor is chosen by the BG-agreement in $R_{m-1,2}$.

In the latter case, there necessarily exists at least one simulator sim_i which proposes that nobody be chosen in that BG-agreement; i.e., simulator sim_i observes no other simulator sim_q with $FinalFaulty_{m-1,q} \geq f$. Since sim_i finished all the BG-agreements in $R_{m-1,1}$, no simulator sim_q with $FinalFaulty_{m-1,q} \geq f$ imposed its proposal in any BG-agreement of $R_{m-1,1}$. Consequently, no processors receive messages from at most a set $FinalFaulty_{m-1,j} < f$ for some simulator sim_j. Since there are now more than

f faulty processors, the set of faulty processors at the beginning of round m must contain a processor from which all messages are received by correct processors. This processor is chosen to be p_l by all the simulators (ties broken by the lowest processor identifier in the case of two such processors).

In both cases, p_l is a correct processor, and we may add or withdraw enough messages to p_l from other faulty processors in the simulation to get exactly f failures. (Every simulator can do that in the same deterministic way.) This is possible since at the beginning of round $m-1$ there are at most f failures.

As round m is the first round in which we choose a correct processor to fail in the second asynchronous phase $R_{m-1,2}$, there are at most k failures per round until round m, as a result of asynchronous simulators being late in a phase. Processor p_l has to decide at the beginning of m exactly when we fail p_l. Its decision may now be read by all the simulators, which can decide on the same value. This concludes the simulation.

Notice that proposing and choosing a correct processor in one of the second asynchronous phases $R_{r,2}$, in order to simulate its failure, is a transient phenomenon, as a result of the second condition in the choice of p_l. Eventually no processor will be proposed to be faulty after round s for s large enough (in fact, in the case when the number of failures is greater than f, then $s = m + 1$). Thus, if a simulator is forever late, then eventually the number of failures in each round is less than k since failures occur only because of asynchrony of simulators, and less than $k+1$ simulators proceed thereafter in the simulation. □

Part 3: *Starting the simulation.* We make the following claim.

CLAIM 4.4. *The algorithm in Figures* 4.3 *to* 4.7 *simulates round* 0 *of synchronous algorithm* $C(k, f)$.

Proof. To start the simulation, each simulator proposes in a series of BG-agreements, one for each processor, its simulator identifier as the value proposed by this processor in code $C(k, f)$. Following these BG-agreements, a simulator starts $R_{1,1}$. The initial state of a processor is determined when the corresponding BG-agreement is resolved. □ □

5. The case of consensus. If we substitute k with 1 in Theorem 4.1, we obtain Theorem 5.1, in which consensus [5, 15] is the same problem as 1-set agreement.

THEOREM 5.1. *For any integer* $f \geq 0$, *no synchronous algorithm* $C(f)$ *solves consensus under the following conditions:*

1. *In runs in which eventually no processors fail in each round, eventually every correct processor decides.*
2. *A processor that sees* f *failures for some fixed* f *decides at the latest after* $f + 1$ *rounds.*

Property 2 is as in the theorem proved in [15]. Property 1 is, however, different: we express our lower bound with no mention of the total number of processes. Indeed, consensus is specified by a termination property which says that eventually every correct processor decides. Even though the property is formulated independently of the actual computation of the run, and because there are finitely many processes and finitely many of them may crash, it is guaranteed that eventually no processor will crash anymore.

Hence (i) correct processors must eventually decide, but (ii) independently of what they decide, we can make the run violate agreement. In [15], it is shown that if a processor that sees f crashes decides at the end of round $f + 1$, then other processors may eventually violate agreement. Indeed it is possible for both 0 and 1

to be the decision value decided by the processor that has seen f crashes. If this processor crashes immediately after deciding, then the other processors have no way to determine which is the decided value.

6. Concluding remarks. Set agreement, in which processors' decisions constitute a set of outputs, is notoriously harder to analyze than consensus, in which the decisions are restricted to a single output. This is because the topological questions that underlie set agreement are not about simple connectivity as in consensus. Analyzing set agreement inspired the discovery of the relation between topology and distributed algorithms and consequently the impossibility of asynchronous set agreement.

Yet, the application of topological reasoning has been to the static case—that of asynchronous and synchronous tasks. It is not known yet, for example, how to characterize starvation-free solvability of nonterminating tasks. Nonterminating tasks are dynamic entities with no defined end. In a similar vein, early deciding synchronous set agreement, in which the number of rounds it takes a processor to decide adapts to the actual number of failures, falls within this category of dynamic entities.

This paper develops a simulation technique that brings to bear topological results to deal with the dynamic situation that arises with early decisions. The novelty of the new simulation is the ability of simulators to look back at the transcript of past rounds of the simulation to influence their current behavior.

Using our new technique, we not only rederive past results (consensus), but we also propose and prove a tight bound on synchronous early deciding set agreement. Our technique uses the BG-simulation in the most creative way to date to obtain a rather simple reduction from a static asynchronous impossibility. This reduction is an alternative to a yet unknown topological argument and in fact may suggest the way of finding such an argument.

REFERENCES

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. ACM, 40 (1993), pp. 873–890.

[2] Y. Afek, G. Stupp, and D. Touitou, *Long-lived and adaptive atomic snapshot and immediate snapshot*, in Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC'00), ACM, New York, 2000, pp. 71–80.

[3] E. Borowsky and E. Gafni, *Generalized FLP impossibility result for t-resilient asynchronous computation*, in Proceedings of the 25th ACM Symposium on Theory of Computing (STOC'93), ACM, New York, 1993, pp. 91–100.

[4] E. Borowsky, E. Gafni, N. Lynch, and S. Rajsbaum, *The BG distributed simulation algorithm*, Distributed Computing, 14 (2001), pp. 127–146.

[5] B. Charron-Bost and A. Schiper, *Uniform consensus is harder than consensus*, J. Algorithms, 51 (2004), pp. 15–37.

[6] S. Chaudhuri, *More choices allow more faults: Set consensus problems in totally asynchronous systems*, Inform. and Comput., 105 (1993), pp. 132–158.

[7] S. Chaudhuri, M. Herlihy, N. Lynch, and M. Tuttle, *Tight bounds for k-set agreement*, J. ACM, 47 (2000), pp. 912–943.

[8] S. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the 3rd ACM Symposium on Theory of Computing (STOC'71), ACM, New York, 1971, pp. 151–158.

[9] D. Dolev, R. Reischuk, and H. R. Strong, *Early stopping in Byzantine agreement*, J. ACM, 37 (1990), pp. 720–741.

[10] M. J. Fischer and N. A. Lynch, *A lower bound for the time to assure interactive consistency*, Inform. Process. Lett., 14 (1982), pp. 183–186.

[11] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.

[12] E. Gafni, *Round-by-round fault detector—unifying synchrony and asynchrony*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC'98), ACM, New York, 1998, pp. 143–152.

[13] M. Herlihy, S. Rajsbaum, and M. Tuttle, *Unifying synchronous and asynchronous message-passing models*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC'98), ACM, New York, 1998, pp. 133–142.
[14] M. Herlihy and N. Shavit, *The topological structure of asynchronous computability*, J. ACM, 46 (1999), pp. 858–923.
[15] I. Keidar and S. Rajsbaum, *On the cost of fault-tolerant consensus when there are no faults—a tutorial*, SIGACT News, Distributed Computing Column, 32 (2001), pp. 45–63.
[16] N. A. Lynch, *Distributed Algorithms*, Morgan–Kaufmann, San Francisco, CA, 1996.
[17] M. Saks and F. Zaharoglou, *Wait-free k-set agreement is impossible: The topology of public knowledge*, in Proceedings of the 25th ACM Symposium on Theory of Computing (STOC'93), ACM, New York, 1993, pp. 101–110.