SIAM J. COMPUT.
Vol. 29, No. 5, pp. 1449–1483

WAIT-FREE k-SET AGREEMENT IS IMPOSSIBLE: THE TOPOLOGY OF PUBLIC KNOWLEDGE*

MICHAEL SAKS† AND FOTIOS ZAHAROGLOU‡

Abstract. In the classical consensus problem, each of n processors receives a private input value and produces a decision value which is one of the original input values, with the requirement that all processors decide the same value. A central result in distributed computing is that, in several standard models including the asynchronous shared-memory model, this problem has no deterministic solution. The k-set agreement problem is a generalization of the classical consensus proposed by Chaudhuri [*Inform. and Comput.*, 105 (1993), pp. 132–158], where the agreement condition is weakened so that the decision values produced may be different, as long as the number of distinct values is at most k. For $n > k \geq 2$ it was not known whether this problem is solvable deterministically in the asynchronous shared memory model. In this paper, we resolve this question by showing that for any $k < n$, there is no deterministic wait-free protocol for n processors that solves the k-set agreement problem. The proof technique is new: it is based on the development of a topological structure on the set of possible processor schedules of a protocol. This topological structure has a natural interpretation in terms of the knowledge of the processors of the state of the system. This structure reveals a close analogy between the impossibility of wait-free k-set agreement and the Brouwer fixed point theorem for the k-dimensional ball.

Key words. distributed computing, consensus, Sperner's lemma, wait-free

AMS subject classifications. 68Q22, 68R10, 54A99

PII. S0097539796307698

1. Introduction.

1.1. Wait-free algorithms and the k-set agreement problem. In totally asynchronous multiprocessor systems without global clocks, the execution speed of each processor may fluctuate widely. A highly desirable property for protocols in such a system is that no processor ever wait indefinitely for an action by another processor, that is, unless a processor fails (stops running) it is guaranteed to complete its task regardless of the relative speeds of the other processors, even if other processors stop participating. Protocols with this property are said to be *wait-free*.

We are interested in the standard model of shared-memory distributed systems with atomic registers [20]; an essentially equivalent model has been studied as *asynchronous parallel random access machines (PRAMs)* (e.g., [12, 22]). We restrict consideration to the case where each processor is deterministic. Informally such a system consists of a set of processors each with its own local memory accessible only to itself, and a set of shared registers. Each shared register supports atomic read and write operations, which means that (1) if two processors access a register simultaneously, the register automatically serializes the accesses, so there are no collisions, and (2)

*Received by the editors August 2, 1996; accepted for publication (in revised form) June 9, 1999; published electronically March 15, 2000. This work was supported in part by NSF grants CCR-8911388, CCR-939215293, and CCR-9700239 and by DIMACS. A preliminary version appeared in the Proceedings of the 24th annual ACM Symposium on Theory of Computing. The material in this paper appeared in somewhat different form in the second author's Ph.D. dissertation, completed at University of California San Diego, May 1993.

http://www.siam.org/journals/sicomp/29-5/30769.html

†Department of Mathematics, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854-8019 (saks@math.rutgers.edu).

‡Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA 92093. This work was performed while the second author was visiting DIMACS.

a processor cannot simultaneously both read and write to a register. A protocol is defined by a set of programs, one for each processor, where each program involves "private computations," together with reads and writes to the shared memory. The system is completely asynchronous and the protocol makes no reference to a clock.

In executing a particular protocol the system may exhibit a wide range of behavior, depending on the relative speeds of the processors. The execution thus depends on the *schedule* of the processors, i.e., the way in which the program steps of the individual processors are interleaved. For a protocol to be correct, it should be correct for *all* schedules. A useful way to think about this requirement is to view the schedule as being chosen by an adversary who seeks to force the protocol to behave incorrectly.

Effective computation in such systems requires some coordination among the processors. The *consensus* problem was introduced as an abstraction of one coordination problem. In this problem, each processor p receives a private input x_p and must produce as output a *decision value* d_p, subject to the following requirements:

Validity. Each decision value is the input value of some processor.

Consistency. All processors that decide must decide the same value.

A fundamental result for the deterministic shared memory model outlined above is that there is no wait-free protocol that solves the consensus problem. This result was proven for this model by Herlihy [16] and independently by Loui and Abu-Amara [21] by adapting the proof of the seminal impossibility result for consensus in message passing systems with one failing processor proved by Fischer, Lynch, and Paterson [15].

This impossibility result spawned considerable activity along several fronts. One direction is to strengthen the model (i.e., introduce randomization, strengthen the shared memory primitives) so as to make consensus achievable. A second direction, pioneered by Herlihy, is the classification of data objects according to the number of processors that can achieve consensus using this data object [16]. A third direction has been to understand fully what can and can't be done in the deterministic atomic register shared-memory model.

As a step toward this third goal, it is natural to consider a weaker version of the consensus problem called the *k-set agreement problem.* This problem is identical to the consensus problem except that the *consistency* condition is replaced by a weaker condition:

k-Consistency. The set of decision values produced by the processors has cardinality at most k.

This problem was proposed and studied by Chaudhuri [10], who considered it in the message passing model and obtained some results relating the difficulty of this problem to various other related problems. In the shared atomic register model the main question is, For which values of n and k is there a wait-free algorithm for n processors to achieve k-set agreement in the shared-memory model? Trivially such an algorithm is possible for $n \leq k$ and (by the result for consensus) is impossible for $k = 1$ and $n > 1$. Chaudhuri conjectured that k-set agreement is impossible for any $n > k$. To appreciate the deceptive difficulty of the problem, the reader may consider the first previously unsolved case $k = 2$ and $n = 3$. This seemingly elementary brain teaser is already quite challenging.

1.2. Main results. In this paper we prove the following theorem.

THEOREM 1.1. *For $k < n$, there is no deterministic wait-free protocol in the shared atomic registers model which solves the k-set agreement problem in a system of n processors.*

The first step in the proof is the formulation of a new formal model for shared-memory atomic register systems, called the *weakly synchronous model*. For our purposes, this model is at least as powerful as the standard ones (so impossibility proofs for this model apply to those) but it has the advantage of a particularly simple combinatorial structure. This allows us to reformulate the given problem in purely combinatorial terms.

We then develop a new approach for reasoning about computability issues in distributed systems. The basis of this approach is to shift focus from the structure of protocols for a distributed system to the structure of the set of possible schedules of a distributed system. To accomplish this shift for a given protocol, we fix ("hardwire") a particular set of input values to the processors and observe that having done this, the processor schedule now completely determines the output values of the processors and thus can be viewed as the "input" to the system. We introduce two key notions: (i) two schedules are "indistinguishable" if for any protocol they exhibit the same output behavior, (ii) a set S of schedules is "knowable" if there is a protocol which "recognizes" it, in the sense that for some specified output symbol, the protocol produces that symbol during an execution if and only if the execution proceeds according to some schedule from S.

These two concepts lead naturally to the definition of a topology on the set of schedules, and Theorem 1.1 is proved by analyzing this topology. Our approach reveals and exploits a close analogy between the impossibility of wait-free k-set agreement and a lemma of Knaster, Kuratowski, and Mazurkiewicz (KKM lemma) [1], which is equivalent to the fixed point theorem for the closed unit ball B_m in m-dimensional Euclidean space: if f is a continuous map from B_m to itself, then there exists a point $x \in B_m$ such that $f(x) = x$. Very roughly, f corresponds to a distributed protocol Π, and the fixed point x corresponds to the schedule for which Π fails to solve the k-set agreement. The increase in difficulty of the k-set agreement proof in going from the case $k = 1$ to the case $k > 1$ corresponds to the increase in difficulty in going from the fixed point theorem for the interval $[-1, 1]$, which is very simple, to the theorem for balls in higher dimension, which, while elementary, is considerably harder. An additional obstacle in our work is that, while the topological structure of B_m is well understood, we must develop the topological structure for the set of schedules from scratch.

While the explicit use of topology can be avoided, we have retained the topological structure of the proof, because this is what drove the proof and it provides important insight into what is going on. Our topological structure has an intuitive interpretation in terms of the information about an execution which is "public knowledge." We believe that it will be worthwhile to explore the connection with the formal theory of distributed knowledge [14].

The inspiration for the topological approach came from Chaudhuri's work [10], in which the combinatorial properties of triangulations in $\mathbf{R}^k$ were used to obtain certain reductions among various decision problems. There is a considerable literature concerning topologies underlying computation in general [27] and distributed computing in particular [26]. The main body of this work seems to center on the use of topology as a tool for describing various semantic constructs in distributing computing, rather than as a tool for proving impossibility results.

Two other research teams—Borowsky and Gafni [7] and Herlihy and Shavit [17]—independently discovered the topological approach to proving impossibility theorems and proved Theorem 1.1. Borowsky and Gafni [7] considered a class of protocols that

is similar to our *weakly synchronous* model, and Herlihy and Shavit [17] considered *full information* protocols that bear similarities to both. The proof of Borowsky and Gafni [7] is similar to ours but does not make the explicit connection to topology. By an additional computational reduction they extend the result to prove the impossibility of k-set agreement for n processors in the presence of k faults. Herlihy and Shavit [17] developed a more general technique for proving impossibility results in this model based on simplicial homology theory. Subsequent developments along these lines include the papers [8, 11, 18].

This paper is organized as follows. In section 2, we formalize the model, the problem, and the above-mentioned notions of indistinguishable schedules and knowable sets. We also present a proof of the impossibility of (1-set) consensus using the new terminology. In section 3, we use this notation to reformulate Theorem 1.1, and we observe a close analogy between this reformulation and the aforementioned KKM lemma. In section 4, we provide an explicit bijection between the set of 2-processor schedules and the points of the closed unit interval that provides an alternative, albeit more complicated, proof of the impossibility of 2-processor consensus, and we sketch an (unproved) correspondence between n-processor schedules and the n-vertex simplex. This informal sketch motivates the combinatorial constructions discussed later. Section 5 contains an outline of the steps that we will follow to emulate the proof of the KKM lemma. Section 6 contains the proof of the main theorem. Some additional facts about the underlying topological structure in our proof are given in an appendix.

2. Definitions and preliminary results.

2.1. Input-output problems and k-set agreement. We fix, once and for all, the set $P = \{1, 2, \ldots, n\}$ of processors. The k-set agreement problem for P is a special case of a larger class of *input-output* problems [23, 6, 5]. In an input-output problem, each processor receives an input value from some set I and must produce an output value from some set D, where the output values must satisfy certain restrictions depending on the input. Formally, such a problem is specified by the input set I, output set D, and a relation $R \subset I^n \times D^n$.

For the k-set agreement problem, we take $I = D = \mathbf{N}$, the set of natural numbers, and the relation consists of pairs $(\vec{x}, \vec{d})$ satisfying the following: The set of values appearing in $\vec{d}$ has size at most k and is a subset of the set of values appearing in $\vec{x}$.

2.2. Informal description of the model. The results in this work will be proved for a specific formalization of the general model of asynchronous distributed computing described in the introduction. For reasons that will be apparent, we call this model the *weakly synchronous model*. This formalization was chosen because it is technically simple and is well suited to formal impossibility proofs. Informally the features of the model are as follows.

- WS.1 Each register is a single-writer–multiple-reader register. The unique processor who may write to a register is referred to as the *owner* of the register.
- WS.2 We assume that each process can simultaneously write to all of the registers it owns in a single atomic step. We model this by having each processor own only one register, whose set of allowed values is an arbitrary infinite set. Thus an arbitrary amount of information can be encoded in a single write.
- WS.3 Each register holds an ordered list of values. When a value v is written to the register it is appended to the list rather than overwriting the existing values. Thus the register contains a record of all the writes ever done to it. At all

times, the length of this list is equal to the total number of write operations that have been performed by the owner.

WS.4 When a processor performs a read operation, it reads the entire shared memory in one atomic step.

WS.5 Each processor's program consists of an infinite loop. Each iteration of the loop consists of a write to its shared register, followed by a read of the entire shared memory, followed by some arbitrary private computation.

WS.6 In solving a decision problem each processor p starts with a private initial input x_p from some input domain I. Each processor must write its decision value in the shared register that it owns. If D is the set of possible decision values, then the first element of D (if any) that the processor writes is considered to be its decision.

WS.7 The system satisfies a weak synchronicity condition. The processors execute their programs in a sequence of synchronous rounds. For each round the scheduling adversary chooses an arbitrary nonempty subset of processors to be active and each active processor executes one iteration of its loop. Since the round is synchronous all active processor writes are completed before any read begins.

We make the informal claim that our model is at least as powerful as other shared-memory models (e.g., [20, 16, 5]). It has been shown in [25] and [19, 13] that restricting to single-writer registers does not reduce the power of the model. Intuitively features 2, 3, and 4 provide more power than the standard models. Features 5 and 6 provide a "normal form" for protocols that solve input-output problems; a program in some other model can easily be converted to one of this form. Feature 7 restricts the power of the adversary by limiting the possible behaviors of the system. Note that this makes it *easier* for a protocol to be correct and thus *harder* for there to be an impossibility proof. We will not present a formal justification of these claims; the interested reader can do this for a favorite model.

2.3. Formal description of the model. A *protocol* Π for a processor set $P = \{1, \ldots, n\}$ and an input domain I is referred to as a (P, I)-protocol and it is specified by a tuple (S, V, e, w, u), where

(1) S is an arbitrary set. S corresponds to the set of possible states for each processor.

(2) V is an arbitrary set. V corresponds to the set of possible values that a processor can write in a register. Thus by WS.2 each register holds an element of V^*, which is the set of finite lists of elements of V.

(3) e is a map from $I \times P$ to S. It determines the initial state of each processor from the processor's input value and the processor's ID.

(4) w is a map from S to V. It determines the value a processor will write in its register on the next step.

(5) u is a map from $S \times (V^*)^n$ to S. It determines the next state of a processor after executing a read operation. The evaluation of u corresponds to an arbitrary private computation by a processor.

A *system configuration* is a pair $(\vec{s}; \vec{l})$, where $\vec{s} = (s_1, \ldots, s_n) \in S^n$ is called the *state configuration* and $\vec{l} = (l_1, \ldots, l_n) \in (V^*)^n$ is called the *memory configuration*. Here s_i is the state of the ith processor and $l_i = v_1; \ldots; v_k \in V^*$ is the list stored in the ith register.

A *block* J is a nonempty subset of P. The *configuration update operator* $\lhd = \lhd_\Pi$ takes as operands a system configuration and a block $J \subset P$ and produces a system

configuration as follows:

$$(\vec{t}, \vec{m}) = (\vec{s}, \vec{l}) \lhd J,$$

where

$$\begin{array}{rcll} m_i & = & l_i & \text{if } i \notin J, \\ m_i & = & l_i, w(s_i) & \text{if } i \in J, \\ t_i & = & s_i & \text{if } i \notin J, \\ t_i & = & u(s_i, \vec{m}) & \text{if } i \in J. \end{array}$$

The operator corresponds to the modification of the system configuration which occurs after the synchronous execution of a program loop by the set J of processors. This corresponds to condition WS.7 in the informal description.

A *schedule* is an infinite sequence of blocks $\sigma = \sigma_1\sigma_2\sigma_3\ldots$. For a nonempty $J \subseteq P$, $\Sigma(J) = \{\sigma = \sigma_1\sigma_2\ldots | \sigma_i \subseteq J, \sigma_i \neq \emptyset\}$ denotes the set of all schedules whose blocks are subsets of J. If $p \in \sigma_i$ we say that processor p takes a step at time i. If $p \in P$ and σ is a schedule, then $steps_p(\sigma)$ is equal to the (possibly infinite) number of steps that p takes in σ. We say that p is

- *active* in σ if $steps_p(\sigma) \geq 1$, i.e., p appears in at least one block. Active(σ) is the set of active processors.
- *inactive* in σ if $steps_p(\sigma) = 0$, i.e., p appears in no block. Inactive(σ) is the set of inactive processors.
- *nonfaulty* in σ if $steps_p(\sigma)$ is infinite. Nonfaulty(σ) is the set of nonfaulty processors.
- *faulty* in σ if $steps_p(\sigma)$ is finite. Faulty(σ) is the set of faulty processors.

Observe that $\Sigma(J)$ consists of those schedules whose set of active processors is a subset of J.

A schedule is always an infinite sequence of blocks. A finite sequence of blocks is called a *fragment*. $\Phi(J)$ denotes the set of fragments whose blocks are subsets of J. The above definitions of $steps_p(\sigma)$, active, and inactive can be extended to fragments, but faulty and nonfaulty make sense only for schedules. If τ is a fragment and ϕ is a schedule or fragment, then $\tau\phi$ represents their concatenation and is a schedule or fragment. If $\sigma = \tau\phi$ we say that τ is a *prefix* of σ or that σ is an *extension* of τ.

If J is a subset of P, we denote by $[J]$ the schedule whose blocks are all equal to J. If J is equal to the singleton set $\{p\}$, we typically write $[p]$ for $[\{p\}]$.

A *run* (resp., *partial run*) is a triple $(\Pi, \vec{x}, \sigma)$, where Π is a (P, I) protocol, $\vec{x} \in I^n$ is the input, and σ is a schedule (resp., a fragment). The *execution* (resp., *partial execution*) $E = E(\Pi, \vec{x}, \sigma)$ associated to a run (resp., partial run) is defined as the infinite (resp., finite) sequence of configurations $C_0C_1C_2\ldots$, where $C_0 = C_0(\Pi, \vec{x}) = (\vec{s}^0, \vec{l}^0)$ is the *initial configuration* defined by $\vec{s}^0 = (e(i_1, 1), \ldots, e(i_n, n))$ and $\vec{l}^0 = (\perp, \ldots, \perp)$, where $\perp$ denotes the empty list, and $C_{i+1} = C_i \lhd \sigma_{i+1}$. The *public record* of the run or partial run $(\Pi, \vec{x}, \sigma)$ is a vector

$$Pub(\Pi, \vec{x}, \sigma) \;=\; (Pub_1(\Pi, \vec{x}, \sigma), \ldots, Pub_n(\Pi, \vec{x}, \sigma)),$$

where $Pub_p(\Pi, \vec{x}, \sigma)$ is the (possibly infinite) list of all writes performed by p in the execution. Note that $Pub_p(\Pi, \vec{x}, \sigma)$ is infinite if and only if σ is a schedule and p is nonfaulty in σ. If $Pub_p(\Pi, \vec{x}, \sigma)$ is finite, then its length is the number of steps that p takes in σ.

Next, we define what it means for a protocol to *compute* a relation $R \subseteq I^n \times D^n$ for some arbitrary set D. The *D-decision value* of p on the run $(\Pi, \vec{x}, \sigma)$, denoted

$d_p^D(\Pi, \vec{x}, \sigma)$, is the first element $d \in D$ that appears on its list, or Λ(null) if none exists. If the element $d \in D$ first appears in the sth item of its list we say that p *D-decides* at step s. The vector of the D-decision values is the *D-decision vector* $\vec{d}^D(\Pi, \vec{x}, \sigma)$.

A vector $\vec{b} \in D^n$ is *compatible* with $\vec{d}$ if it can be obtained from $\vec{d}$ by replacing each Λ by some element of D. An input $\vec{x}$ is *R-permissible* if there is at least one vector $\vec{d} \in D^n$ such that $(\vec{x}, \vec{d}) \in R$. Protocol Π *computes* the relation R on schedule σ if for all R-permissible inputs $\vec{x}$

(1) the D-decision value of every nonfaulty processor is not null,
(2) there is a vector $\vec{b} \in D^n$ with $(\vec{x}, \vec{b}) \in R$ which is compatible with the decision vector $\vec{d}^D(\Pi, \vec{x}, \sigma)$.

A protocol Π is an *f-fault tolerant protocol* for R if it computes R on σ for all σ such that $|\text{Faulty}(\sigma)| \leq f$. A protocol that is $(n-1)$-fault tolerant, i.e., one that computes R on every schedule σ, is said to be *wait-free*. A protocol Π is a *bounded wait-free* protocol for R if there is a B such that for every run, each processor that takes at least B steps D-decides. It is easy to see (and is well known) that a wait-free protocol is bounded wait-free. It is also known (see [9] and the remark following Lemma 6.1 below) that for k-set consensus the existence of a wait-free protocol implies the existence of a bounded wait-free protocol.

Finally, a (P, I) protocol Π is *input-free* if the initial state of each processor depends on the processor ID only, that is, e is a map from P to S instead of from $I \times P$ to S. For an input-free protocol Π we write (Π, σ) for the run or partial run, $E(\Pi, \sigma)$ for the execution or partial execution, and $Pub(\Pi, \sigma)$ for the public record. Intuitively, input-free protocols are obtained from arbitrary ones by "hardwiring" a specific set of inputs to the individual processors.

PROPOSITION 2.1. *Let Π be a (P, I) protocol and let $\vec{x} \in I^n$ be some fixed input vector. Then there exists an input-free protocol Π' such that for all $\sigma \in \Sigma(P) \cup \Phi(P)$,*

$$E(\Pi, \vec{x}, \sigma) = E(\Pi', \sigma)$$

2.4. Impossibility of consensus. For illustration purposes we show how to adapt the proof of the consensus impossibility result [15, 16, 21] to prove that wait-free consensus is impossible in the weakly synchronous model. The previous proofs in the literature give a stronger result: there is no consensus protocol that is even 1-fault tolerant. It is possible to strengthen the following proof to give this result, but we do not do this here. This section is not needed for the development of the rest of the paper.

THEOREM 2.2. *In the weakly synchronous model, for $n > 1$ there is no deterministic wait-free protocol for n-processor consensus.*

Proof. Assume the set of possible inputs is $I = \{a, b\}$. Suppose, for contradiction, that Π is a protocol that solves consensus for all $\vec{x} \in I^n$ and all schedules σ. Each run $(\Pi, \vec{x}, \sigma)$ may be classified uniquely as a-deciding or b-deciding depending on the decision value.

PROPOSITION 2.3. *There is an input vector $\vec{y}$ and two schedules σ and ϕ such that $(\Pi, \vec{y}, \sigma)$ is a-deciding and $(\Pi, \vec{y}, \phi)$ is b-deciding.*

Proof. Let $\vec{y} \in I^n$ be an input vector with the minimum number of a's such that there exists a schedule σ such that $(\Pi, \vec{y}, \sigma)$ is a a-deciding; $\vec{y}$ has at least one a in it, say, in coordinate p. Define ϕ to be the schedule with repeated blocks $P - \{p\}$. We claim that $(\Pi, \vec{y}, \phi)$ is b-deciding. Let $\vec{w}$ be the vector obtained from $\vec{y}$ by changing the entry in coordinate p to b. Both of the runs $(\Pi, \vec{y}, \phi)$ and $(\Pi, \vec{w}, \phi)$ have the same

public record. Furthermore the latter is b-deciding since $\vec{w}$ has fewer a's from $\vec{y}$ which was selected to have the minimum possible such number. Therefore $(\Pi, \vec{y}, \phi)$ is also b-deciding. □

We restrict our attention to runs with input $\vec{y}$ given by Proposition 2.3. We say that a fragment τ is a-valent (resp., b-valent) if for every schedule ρ, $(\Pi, \vec{y}, \tau\rho)$ is a-deciding (resp., b-deciding). It is *bivalent* if it is neither a-valent nor b-valent, i.e., if there are schedules ρ and μ such that $(\Pi, \vec{y}, \tau\rho)$ is an a-deciding and $(\Pi, \vec{y}, \tau\mu)$ is a b-deciding. Clearly, no processor reaches a decision on the partial execution corresponding to a bivalent fragment. By choice of $\vec{y}$, the empty fragment is *bivalent.*

LEMMA 2.4. *For any bivalent fragment τ there exists a block J such that the fragment τJ is bivalent.*

Proof. Let τ be a bivalent fragment. Assume for contradiction that τJ is not bivalent for any block J. Thus for each J, τJ is either a- or b-valent. Without loss of generality suppose that τP (where P is the set of all processors) is a-valent. We will show that τJ is a-valent for every J in P which would imply τ is a-valent, a contradiction. Let J be arbitrary and $\bar{J} = P - J$. It is easy to see that the state of the shared memory and the internal states of processors in $\bar{J}$ are identical for the partial executions $PE(\Pi, \vec{y}, \tau P)$ and $PE(\Pi, \vec{y}, \tau J\bar{J})$. (Note, however, that the internal states of processors in J may differ between the two partial executions.) Recall that $[\bar{J}]$ denotes the schedule consisting of repeated $\bar{J}$ blocks. Then schedules $\tau J\bar{J}[\bar{J}]$ and $\tau P[\bar{J}]$ have identical public records. Since by our assumption τJ is not bivalent it must be a-valent, giving the desired contradiction. □

By using the above lemma, one can construct an (infinite) schedule such that any prefix is bivalent, which contradicts that Π is a wait-free algorithm for consensus. □

2.5. Tallies and the counting protocol. We now introduce a specific protocol, called the *counting protocol,* which will play a special role in our analysis. This is an input-free protocol which we denote by Γ.

To describe this protocol, we need to introduce the notion of a *tally vector*, which is a vector indexed by P whose entries are nonnegative integers. For a tally vector v, and $p \in P$, we write $v[p]$ for the $[p]$ entry of v. We define the partial order on tally vectors with $v \leq w$ if $v[p] \leq w[p]$ for all $p \in P$. Two tally vectors that are comparable under this ordering are said to be *noncrossing* and they are *crossing* otherwise. If τ is a fragment, the *tally* of τ, denoted $t(\tau)$, is the tally vector such that $t(\tau)[q]$ is equal to the number of steps q takes in τ. If $\sigma = \sigma_1, \sigma_2, \ldots$ is a schedule or fragment, the *tally sequence* associated with σ, denoted $\mathrm{T}(\sigma)$, is the sequence $T_0, T_1, T_2, \ldots$ of tally vectors, where T_i is the tally of the fragment $\sigma_1\sigma_2 \ldots \sigma_i$. For example, the fragment $\{1,2\}\{3\}\{1,3\}\{1,2,3\}\{1\}\{2\}$ has tally sequence $(0,0,0), (1,1,0), (1,1,1), (2,1,2), (3,2,3), (4,2,3), (4,3,3)$. Observe that the tally vectors in the tally sequence of σ are noncrossing and that σ is trivially determined by $\mathrm{T}(\sigma)$.

We can now define the counting protocol Γ. As stated before, it takes no input. The state of each processor p is a tally vector $tally_p$. Initially all entries of $tally_p$ are 0. Each time p executes a step, p writes (appends) $tally_p$ to its public register. It then reads all of the shared registers and sets $tally_p$ so that $tally_p[q]$ is equal to the number of steps that have been taken by processor q (which is equal to the length of the list in processor q's public register). We define $Count(\sigma)$ to be the public record, $Pub(\Gamma, \sigma)$, of the counting protocol on schedule or fragment σ and call this the *public tally* of σ. Thus for each $p \in P$, $Count_p(\sigma)$ is a (possibly infinite) list of length $steps_p(\sigma)$,

where each element of the list is itself a vector indexed by P. For $i \leq steps_p(\sigma)$, we denote by $Count_{p,i}(\sigma)$ the vector corresponding to the ith write by p, and for $q \in P$, $Count_{p,i}(\sigma)[q]$ is the value of this vector in position q. For example, on the fragment $\{1,2\}\{3\}\{1,3\}\{1,2,3\}\{1\}\{2\}$, we have

$$Count_1(\sigma) = (0,0,0); (1,1,0); (2,1,2); (3,2,3),$$
$$Count_2(\sigma) = (0,0,0); (1,1,0); (3,2,3),$$
$$Count_3(\sigma) = (0,0,0); (1,1,1); (2,1,2).$$

Note that the ith write by processor p is $t(\tau)$, where τ is the prefix up to the $(i-1)$st step of p. Note also that the final values of the internal states of the processors, $tally_1$, $tally_2$, and $tally_3$, are, respectively, $(4,2,3)$, $(4,3,3)$, and $(3,2,3)$, which do not appear in the public record.

By definition, each tally that appears in the public tally of σ also appears in its tally sequence $\mathrm{T}(\sigma)$. In particular, the set of all tally vectors that appear in the public tally is noncrossing. We will need the following lemma.

LEMMA 2.5. *Let σ and ϕ be two schedules such that $Count(\sigma) \neq Count(\phi)$. Then at least one of the following holds:*

1. *There exists a processor p and an integer i such that p takes at least i steps in both σ and ϕ and the tally vectors $Count_{p,i}(\sigma)$ and $C_{p,i}(\phi)$ are different.*
2. *There exists a pair of crossing tally vectors v and w so that v appears in $Count(\sigma)$ and w appears in $Count(\phi)$.*

Proof. First consider the case that each processor takes exactly the same number of steps in σ as in ϕ. Then since $Count(\sigma) \neq Count(\phi)$, the first conclusion must hold.

Next, suppose that some processor p takes a total of i steps in σ and takes at least $i+1$ steps in ϕ. Let r be a processor that takes infinitely many steps in ϕ and let w be a tally vector written by r on schedule ϕ such that $w(p) \geq i+1$. Such a w must exist since p takes at least $i+1$ steps and r takes infinitely many steps. Let q be a processor that takes infinitely many steps in σ and let v be the tally vector written by q on schedule σ at its $w(q)+2$ step. Then $v(q) = w(q)+1$ and $v(p) \leq i < w(p)$, and so v and w are crossing tally vectors. ∎

2.6. Indistinguishable schedules. Two schedules or fragments σ and τ are *publicly indistinguishable* if for any protocol Π and input vector $\vec{x}$, the public records $Pub(\Pi, \vec{x}, \sigma)$ and $Pub(\Pi, \vec{x}, \tau)$ are the same. Intuitively, this says that there is no protocol Π and input $\vec{x}$ that will enable an "outside observer" looking at the "final" lists in the registers to distinguish between the schedules σ and τ. This is clearly an equivalence relation on the set all schedules and fragments. The structure of this equivalence relation is fundamental to the proofs of our results.

If σ and τ are publicly indistinguishable, then they must have the same public tallies, i.e., $Count(\sigma) = Count(\tau)$. As we will see in Theorem 2.12, this condition is also sufficient for public indistinguishability. This theorem will also provide another combinatorial characterization based on a notion called *compression*. To introduce this notion we will need some additional definitions.

If τ is a fragment, its length $|\tau|$ is the number of blocks in it, and its *weight*, $w(\tau)$, is the sum of the block sizes. Associated to each schedule or fragment σ is its *site sequence* $s(\sigma) = (s_1, s_2, \ldots)$ of length $|\sigma|$, where s_i is the weight of the first i blocks. We will also refer to s_i as the *site* of the ith block of σ.

EXAMPLE 2.6. *$P = \{1,2,3\}$ and τ is the fragment $\{1,2,3\}\{1,2\}\{2,3\}\{2\}\{3\}$. Then $|\tau| = 5$, $w(\tau) = 9$, and $s(\tau) = (3,5,7,8,9)$.*

Let σ be a schedule or fragment. Then a block σ_i of σ is *hidden* if σ_i is not the last block and no processor in σ_i appears in any later block. Note that any two hidden blocks are necessarily disjoint and that whether σ is a schedule or a fragment, at least one processor belongs to no hidden block. Thus we have the following proposition.

PROPOSITION 2.7. *A schedule or fragment σ has at most $|P|-1$ hidden blocks.*

A schedule or fragment that has no hidden blocks is said to be *compressed*.

We now define operators for "eliminating" hidden blocks. For a positive integer s, the *merge* operator M_s is defined as follows. If σ is a schedule or fragment, $M_s(\sigma)$ is the schedule or fragment obtained as follows: if there is a block σ_i at site s and the block is hidden, then replace σ_i and σ_{i+1} by their union; otherwise, $M_s(\sigma)=\sigma$. The following facts are easy to verify.

PROPOSITION 2.8.

(1) *If σ is compressed, then $M_s(\sigma)=\sigma$ for all s.*
(2) *The operators M_s and M_r commute for all integers r and s.*
(3) *If σ is a schedule or fragment and $r_1, r_2, \ldots, r_k$ are the sites of its hidden blocks, then $M_{r_1}M_{r_2}\ldots M_{r_k}(\sigma)$ is a compressed sequence.*

The compressed sequence obtained from σ in the third part of Proposition 2.8 is called the *compression* of σ and is denoted $\hat{\sigma}$. More generally, a sequence τ which can be obtained from σ by application of some sequence of merge operators is said to be a *partial compression* or σ. An easy consequence of Proposition 2.8 is the following.

COROLLARY 2.9. *If τ is a partial compression of σ, then $\hat{\tau}=\hat{\sigma}$. Thus $\hat{\sigma}$ is the unique compressed sequence that can be obtained from σ by applying merge operators.*

The compression map $\sigma \longrightarrow \hat{\sigma}$ defines an equivalence relation on $\Sigma(P)\cup\Phi(P)$: σ and τ are *compression equivalent* if $\hat{\sigma}=\hat{\tau}$. The equivalence class of σ is called the *compression class* of σ and is denoted $\langle\sigma\rangle$.

Let σ be a compressed schedule, let χ be the smallest prefix of σ containing all faulty processors, and write $\sigma=\chi\phi$. Then any schedule that compresses to σ is of the form $\tau\phi$, where τ is a fragment of the same weight as χ. In particular, this implies the following.

PROPOSITION 2.10. *The compression class $\langle\sigma\rangle$ of any schedule is finite.*

EXAMPLE 2.11. *Suppose $P=\{1,2,3,4\}$ and let σ be the compressed schedule $\{1,2\}\{1,3,4\}\{1,2,3\}[3]=\{1,2\}\{1,3,4\}\{1,2,3\}\{3\}\{3\}\ldots$. There are eleven uncompressed schedules whose compression is σ:*

$$\begin{aligned}
\sigma^1 &= \{1,2\}\{1,3,4\}\{1\}\{2,3\}[3],\\
\sigma^2 &= \{1,2\}\{1,3,4\}\{2\}\{1,3\}[3],\\
\sigma^3 &= \{1,2\}\{1,3,4\}\{1\}\{2\}[3],\\
\sigma^4 &= \{1,2\}\{1,3,4\}\{2\}\{1\}[3],\\
\sigma^5 &= \{1,2\}\{1,3,4\}\{12\}[3],\\
\sigma^6 &= \{1,2\}\{4\}\{1,3\}\{1,2,3\}[3],\\
\sigma^7 &= \{1,2\}\{4\}\{1,3\}\{1\}\{2,3\}[3],\\
\sigma^8 &= \{1,2\}\{4\}\{1,3\}\{2\}\{1,3\}[3],\\
\sigma^9 &= \{1,2\}\{4\}\{1,3\}\{1\}\{2\}[3],\\
\sigma^{10} &= \{1,2\}\{4\}\{1,3\}\{2\}\{1\}[3],\\
\sigma^{11} &= \{1,2\}\{4\}\{1,3\}\{1,2\}[3].
\end{aligned}$$

In σ^{10}, for instance, the hidden blocks are at sites 3, 6, *and* 7. *Applying M_6 yields σ^{11}, and then applying M_7 yields σ^6, and applying M_3 yields σ.*

The following result characterizes public indistinguishability. Recall the definition of the public tally vector $Count(\sigma)$ from the previous section as the public record of the counting protocol Γ.

THEOREM 2.12. *Let σ and τ be schedules or fragments. Then the following are equivalent:*

(1) *σ is publicly indistinguishable from τ,*
(2) *$Count(\sigma) = Count(\tau)$,*
(3) *$\hat{\sigma} = \hat{\tau}$.*

Proof. (3) $\Rightarrow$ (1). Assume that $\hat{\sigma} = \hat{\tau}$; we will show that σ and τ are publicly indistinguishable.

LEMMA 2.13. *σ is publicly indistinguishable from $M_s(\sigma)$ for all s.*

Proof. The result is trivial if $\sigma = M_s(\sigma)$, so assume they are distinct. Then σ has a hidden block σ_k at site s and

$$\begin{array}{rcll} \phi_i & = & \sigma_i & \text{if } i < k, \\ \phi_k & = & \sigma_k \cup \sigma_{k+1}, & \\ \phi_i & = & \sigma_{i+1} & \text{if } i > k. \end{array}$$

Let $C(\sigma, i)$ be the configuration of the protocol Π on schedule σ after the execution of i blocks. Clearly $C(\phi, i) = C(\sigma, i)$ for every $i < k$. The configuration $C(\sigma, k+1)$ and $C(\phi, k)$ have exactly the same memory configuration but they may differ on the state configuration of the processors in the set σ_k. But these processors will not execute another step later in any of the schedules. Therefore, for $i > k$, $C(\phi, i)$ differs from $C(\sigma, i+1)$ only in the state of the processors in σ_k. Therefore σ and ϕ are publicly indistinguishable. □

COROLLARY 2.14. *σ is publicly indistinguishable from $\hat{\sigma}$.*

Proof. Let $r_1, r_2, \ldots, r_k$ be the sites of the hidden blocks of σ. Then by proposition 2.8, $\hat{\sigma} = M_{r_1} M_{r_2} \ldots M_{r_k}(\sigma)$ and the result follows by applying the previous lemma and induction. □

Therefore σ is publicly indistinguishable from $\hat{\sigma}$ and τ is publicly indistinguishable from $\hat{\tau}$. Since $\hat{\sigma} = \hat{\tau}$, then σ and τ are publicly indistinguishable.

(1) $\Rightarrow$ (2). This is trivial since if σ is publicly indistinguishable from τ, then by definition every protocol, including Γ, has the same public record on σ as τ.

(2) $\Rightarrow$ (3). Let σ and τ be schedules such that $Count(\sigma) = Count(\tau)$. From Corollary 2.14 and the fact that (1) $\Rightarrow$ (2) we have $Count(\sigma) = Count(\hat{\sigma})$ and $Count(\tau) = Count(\hat{\tau})$, hence $Count(\hat{\sigma}) = Count(\hat{\tau})$.

Now suppose for contradiction that $\hat{\sigma} \neq \hat{\tau}$. Write $\phi = \hat{\sigma}$ and $\mu = \hat{\tau}$ and consider the least k such that $\phi_k \neq \mu_k$. Let ϕ' and μ' be the prefixes ending with ϕ_k and μ_k, respectively. Then the tally vectors $t(\phi')$ and $t(\mu')$ are different; we may assume that either $t(\phi') < t(\mu)$ or $t(\phi')$ and $t(\mu')$ are crossing vectors (defined in section 2.5). Then the vector $t(\phi')$ does not appear in the tally sequence of μ and does not appear in $Count(\mu)$. On the other hand, since ϕ is compressed, we may choose a processor q in ϕ_k that writes again. Its next write in the counting protocol will be $t(\phi')$, contradicting that $Count(\phi) = Count(\mu)$.

This completes the proof of Theorem 2.12. □

2.7. Quasi extensions of fragments. We have defined the notion of a public record associated to an execution as the vector $\vec{R}$ indexed by p, where the entry $\vec{R}_p$ is the list of all writes done by p during the execution. It is convenient to call any such

vector $\vec{R}$ indexed by P, where the entry $\vec{R}_p$ is an arbitrary list of elements, a *public record*. A public record is *finite* if each list is finite and infinite otherwise. If $\vec{R}$ and $\vec{R}'$ are public records, then we say that $\vec{R}'$ is an extension of $\vec{R}$ if each of the lists $\vec{R}_p$ is a prefix of the corresponding list $\vec{R}'_p$.

Now if τ is a fragment which is a prefix of the schedule or fragment σ, then clearly the following condition holds:

(QE) For any protocol Π and input $\vec{x}$, the public record of the execution $(\Pi, \vec{x}, \sigma)$ is an extension of the public record of the execution $(\Pi, \vec{x}, \tau)$.

We say that a schedule or fragment σ is a *quasi extension* of the fragment τ if condition (QE) holds. For a fragment τ, Q_τ denotes the set of schedules that are quasi extensions of τ.

Condition (QE) does not imply that τ is a prefix of σ. For instance, we have the following.

LEMMA 2.15. *Let ρ be a fragment, ϕ a schedule or fragment, and $Y \subseteq Active(\phi)$. If $\sigma = \rho\phi$ and $U \subseteq Active(\phi)$, then σ quasi-extends ρU.*

Proof. For any protocol, the public records of $\rho\phi$ and ρU trivially agree up through the end of ρ. Now in ρU, each of the processors in U write once more, and they write the view observed during their last step in ρ. But for each $p \in U$, the same write will occur when executing $\rho\phi$ since $p \in Active(\phi)$. □

We have the following combinatorial characterization of quasi extension, which is analogous to Theorem 2.12.

THEOREM 2.16. *Let μ be a fragment and let σ be a schedule or a fragment. Then the following are equivalent:*

(1) *σ is a quasi extension of μ.*
(2) *$Count(\sigma)$ extends $Count(\mu)$.*
(3) *There exists a prefix ρ of σ, a schedule or fragment ϕ, and a subset U of $Active(\phi)$ such that $\sigma = \rho\phi$ and $\hat{\mu} = \widehat{\rho U}$.*

Proof. (1) ⇒ (2). This follows from the definition of quasi extension and the fact that $Count(\sigma)$ and $Count(\mu)$ are the respective public records arising from a protocol.

(2) ⇒ (3). Write $\hat{\mu}$ as τU, where U is the last block of $\hat{\mu}$. By hypothesis, and the fact that μ is publicly indistinguishable from $\hat{\mu}$, we have that $Count(\sigma)$ extends $Count(\tau U)$. Since τU is compressed, there is a processor $p \in U$ that also appears in the last block of τ. Let i be the number of steps that p makes in τ. By definition of the counting protocol, $Count_{p,i+1}(\tau U) = t(\tau)$, where $t(\tau)$ is the tally vector associated with fragment τ. By hypothesis, $Count_{p,i+1}(\sigma) = Count_{p,i+1}(\tau U)$. Let ρ be the minimal prefix of σ containing the first i steps of p. Again, by the definition of the counting protocol, $Count_{p,i+1}(\sigma) = t(\rho)$. Hence $t(\tau) = t(\rho)$. Write $\sigma = \rho\phi$. Now, since $Count(\rho\phi)$ extends $Count(\tau U)$ it is clear that each processor in U must take a step in ϕ, so $U \subseteq Active(\phi)$. Finally, we claim that $\widehat{\rho U} = \tau U$, and for this it is enough, by Theorem 2.12, to show that $Count(\rho U) = Count(\tau U)$. By Lemma 2.15, $Count(\rho\phi)$ extends $Count(\rho U)$ and since $Count(\rho\phi)$ also extends $Count(\tau U)$ (by hypothesis) and every processor takes the same number of steps in ρU as in τU, we must have $Count(\rho U) = Count(\tau U)$, as required.

(3) ⇒ (1). Assume that (3) holds. By Lemma 2.15, σ is a quasi extension of ρU. But since $\widehat{\rho U} = \hat{\mu}$, ρU and μ are publicly indistinguishable and so σ is also a quasi extension of μ. □

2.8. Knowable sets. For a set of schedules S, let $\hat{S} = \{\hat{\sigma} | \sigma \in S\}$. In particular, $\hat{\Sigma}$ is the set of all compressed schedules. By Theorem 2.12, to check that a protocol

for an input-output problem is correct, it suffices to check it for schedules in $\hat{\Sigma}$.

In this subsection, we are interested in the dependence of the public record of a run $(\Pi', \vec{x}, \sigma)$ on σ while holding Π' and $\vec{x}$ fixed. Therefore it is easier to consider the corresponding input-free protocol Π given by Proposition 2.1.

In an input-free protocol, each processor's decision depends only on the schedule. One can view the computational steps taken by the processors as "collecting information" about the schedule. When a processor "knows enough" about the schedule, it can make a decision. Let $K(\Pi, d)$ be the set of all compressed schedules on which some processor running the input-free protocol Π writes d on its list. The pair (Π, d) is an *acceptor* and d is its *accepting* value. We say that (Π, d) *accepts schedule* σ if $\sigma \in K(\Pi, d)$. A set $K \subseteq \hat{\Sigma}$ is *publicly knowable* or simply *knowable* if it equals $K(\Pi, d)$ for some acceptor (Π, d). Below we give several examples. In each example, $d =$"@."

EXAMPLE 2.17. *$\hat{\Sigma}$ is knowable. If Π is the protocol where every processor writes* "@" *at every opportunity, then $\hat{\Sigma} = K(\Pi, @)$.*

EXAMPLE 2.18. *$\emptyset$ is knowable. If Π is the protocol where every processor writes* "0" *at every opportunity, then $K(\Pi, @) = \emptyset$.*

EXAMPLE 2.19. *For each integer k and processor p, let $S_{p,k}$ be the set of compressed schedules in which processor p takes at least k steps. It is easy to see that $S_{p,k}$ is knowable; a simple input-free protocol that accepts this set is the one where processor p writes* "0" *for its first $k-1$ steps and* "@" *thereafter, and all other processors always write* "0."

EXAMPLE 2.20. *Let S_i be the set of schedules where every processor takes at least i steps. S_i is a knowable set. The protocol that accepts the set is as follows: Each processor has two possible states, "continue" and "accept." While in the "continue" state, each processor appends* "0" *to the list in its shared register, reads the shared memory, and enters the "accept" state if all of the processors took i steps (have output list of length at least i). Once the processor is in the "accept" state, it writes* "@."

EXAMPLE 2.21. *Let τ be any fragment. The set $\hat{Q}_\tau$ of compressed schedules that are quasi extensions of τ is knowable. Define the following modification of the counting protocol defined in section* 2.5*: if any processor ever observes that the public record is an extension of $Count(\tau)$, then it writes* "@" *at its next opportunity. When this protocol is run on an (infinite) compressed schedule σ,* "@" *is written if and only if $Count(\sigma)$ is an extension of $Count(\tau)$. By Theorem* 2.16 *this is equivalent to $\sigma \in \hat{Q}_\tau$.*

For contrast, we give some examples of sets that are not knowable.

EXAMPLE 2.22. *Let $T_{p,k}$ be the set of compressed schedules where processor p takes* exactly *k steps for some k. Then $T_{p,k}$ is* not *a knowable set. Suppose to the contrary that (Π, d) is an acceptor for $T_{p,k}$. Let $\sigma \in T_{p,k}$ be arbitrary; then d is written in the public record of the run (Π, σ). Now the first write of (a, "d") occurs at some finite block σ_m of σ, so if we define $\phi = \sigma_1, \sigma_2, \ldots, \sigma_m, \{p\}, \{p\}, \ldots$, then ϕ is also accepted by (Π, d). Then $\hat{\phi}$ is a compressed schedule accepted by (Π, d) but not in $T_{p,k}$, a contradiction.*

EXAMPLE 2.23. *The complement of a knowable set need not be knowable, e.g., consider the complement of $S_{p,k}$ and apply an argument analogous to the previous one.*

EXAMPLE 2.24. *Let N_p be the set of compressed schedules that do not begin with $\{p\}$. Then N_p is not knowable. Suppose to the contrary that (Π, d) is an acceptor for N_1. Let $\sigma \in N_1$ be the schedule $\{1,2\}, \{2\}, \{2\}, \ldots$. Then there is a block σ_k of σ such that d is first written in the public record of the run (Π, σ). If*

$\phi = \{1\}, \{2\}^k, \{1,2\}\{2\}, \{2\}, \{2\}, \ldots$, *then the public record corresponding to its first* $k+1$ *blocks will match the public record of the first* k *blocks of* σ. *Thus* ϕ *is compressed, is not in* N_1, *and is accepted by* (Π, d), *a contradiction.*

The last example illustrates an important point: it is not hard to construct a protocol such that for any schedule σ in N_p, the fact "$\sigma \in N_p$" is recorded in the local state of at least one processor q. (Each processor $q \neq p$ records "yes" in its local state, if its first read does not see a write by p, and p records "yes" in its local state if its first read sees at least one write other than its own.) However, in this case, if p appears in the first block and never writes again, this fact does not become "public." The term *publicly knowable* reflects the fact that the value v is written in the memory and thus every nonfaulty processor "eventually knows" that the schedule belongs to K.

An acceptor (Π, d) is said to accept a *fragment* τ if d appears in the public record of Π on τ. The definition of a run of a protocol implies the following.

PROPOSITION 2.25. *Let* K *be a knowable set. Then a compressed schedule* σ *belongs to* K *if and only if for some fragment* τ *of* σ, $\hat{Q}_\tau \subseteq K$.

The concept of knowable set allows us to reformulate the impossibility result for a k-set agreement problem. We do this in the next section.

3. Reformulating Theorem 1.1 and a topological analogy. We want to prove that, in the weakly synchronous model, there is no wait-free protocol that solves the k-set agreement problem in a system of n processors for any $k < n$. Clearly it suffices to prove the impossibility result for $k = n - 1$. As a first step in the proof of Theorem 1.1, we use the notation of the previous section to reformulate it.

Assume, for contradiction, that there exists a wait-free protocol Π that solves the $(n-1)$-set agreement problem for a processor set $P = \{1, \ldots, n\}$. Consider the behavior of Π on the input $\vec{x} = (1, \ldots, n) \in I^n$. For each $i \in P$, let D_i be the set of compressed schedules σ such that on the run $(\Pi, \vec{x}, \sigma)$ at least one processor reaches decision i. By definition, each D_i is a knowable subset of $\hat{\Sigma}$. Also, if Π is correct, then for any schedule σ, at least one processor decides some value in $\{1, \ldots, n\}$, so the sets $D_1, D_2, \ldots, D_n$ together cover $\hat{\Sigma}$. An arbitrary sequence $A_1, A_2, \ldots, A_n$ of subsets of $\hat{\Sigma}$ *satisfies the activity property* if for each $p \in P$, processor p is active in each $\sigma \in A_p$. In other words A_p is disjoint from $\hat{\Sigma}(P - \{p\})$.

PROPOSITION 3.1. *If* Π *is a fully fault-tolerant protocol for* k*-set agreement, then the sequence of sets* $D_1, D_2, \ldots, D_n$ *satisfies the activity property.*

Proof. Assume for the sake of contradiction that for some schedule $\sigma \in D_p$, processor p is not active. Consider a run of protocol Π on σ with the input vector $\vec{y}$ defined by $y_p = n + 1$ and $y_q = x_q$ for $q \neq p$. Then $Pub(\Pi, \vec{x}, \sigma) = Pub(\Pi, \vec{y}, \sigma)$; therefore, on input $\vec{y}$ some processor decides p although p does not appear in $\vec{y}$, which violates the validity condition. □

Our main theorem will thus follow from the following general result about knowable sets.

THEOREM 3.2. *If* $K_1, \ldots, K_n$ *is a collection of knowable subsets of* $\hat{\Sigma}(P)$ *that cover* $\hat{\Sigma}$ *and satisfy the activity property, then*

$$\bigcap_{p=1}^{n} K_p \neq \emptyset.$$

Applying this to the sets $D_1, D_2, \ldots, D_n$, we obtain that there is a single schedule σ in which every possible decision $1, 2, \ldots, n$ is reached, contradicting that Π solves the $(n-1)$-set agreement problem.

TABLE 1
The syntactic correspondence between $\mathbf{R}^n$ and knowable set topologies.

$Hull(E^P)$	$\longleftrightarrow$	$\hat{\Sigma}(P)$
point $\vec{z} \in Hull(E^P)$	$\longleftrightarrow$	compressed schedule σ
relatively open subset of $Hull(E^P)$	$\longleftrightarrow$	knowable subset of $\hat{\Sigma}(P)$
$Hull(E^{P-\{p\}})$	$\longleftrightarrow$	$\hat{\Sigma}(P - \{p\})$
boundary property	$\longleftrightarrow$	activity property
vertex $\vec{e}^p$	$\longleftrightarrow$	schedule $[p] = \{p\}\{p\}\ldots$

There is a striking analogy between the statement of Theorem 3.2 and a well-known theorem concerning the topology of Euclidean space. Let $Hull(Z)$ denote the convex hull of a set of points Z in the Euclidean space. Let $E^P = \{\vec{e}^p | p \in P\}$ be the set of standard unit basis vectors of $\mathbf{R}^P$, which we identify with $\mathbf{R}^n$. Note that $Hull(E^P)$ is the $(n-1)$-dimensional simplex consisting of the set of nonnegative vectors whose coordinates sum to 1. We say that a sequence of subsets $A_1, A_2, \ldots, A_n$ of $Hull(E^P)$ *satisfies the boundary property* if for each $p \in \{1, \ldots, n\}$, A_p is disjoint from $Hull(E^{P-\{p\}})$, which is the face of $Hull(E^P)$ opposite the vertex $\vec{e}^p$, i.e, each vector $\vec{z} \in A_p$ has positive pth coordinate. Finally, a subset U is *relatively open* in $Hull(E^P)$ if it is the intersection of $Hull(E^P)$ with an open subset of $\mathbf{R}^n$.

The following theorem is essentially equivalent to the Brouwer fixed point theorem for the $(n-1)$-dimensional closed ball.

THEOREM 3.3 (KKM theorem; see [1]). *If $U_1, \ldots, U_n$ is a collection of relatively open subsets of $Hull(E^P)$ that cover $Hull(E^P)$ and satisfy the boundary property, then*

$$\bigcap_{i=1}^{n} U_i \neq \emptyset.$$

There is a tight syntactic correspondence between the two situations described by Theorems 3.2 and 3.3, which is given in Table 1.

The obvious question is, Is there some way to make use of this correspondence to prove the desired result for knowable sets? The natural way to do this would be to find a bijection between $\hat{\Sigma}(P)$ and $Hull(E^P)$ which obeys the syntactic correspondence. Given such a bijection Theorem 3.2 would follow from Theorem 3.3. In fact, we believe that there is such a bijection and that we have an existential argument for this. But the technical details involved in turning this argument into a rigorous proof seem to be considerable and except for the case $n = 2$, we have not completed such a proof.

It turns out we don't really need such a bijection. We prove Theorem 3.2 directly by analyzing and imitating the proof of Theorem 3.3. Nevertheless, the ideas of the proof are based on the intuition we developed in trying to construct an appropriate bijection between $\hat{\Sigma}(P)$ to $Hull(E^P)$. In the next section, we discuss some of these intuitions and give an explicit bijection for the $n = 2$ case (which corresponds to the impossibility of 2-processor consensus), a not-so-explicit bijection for the $n = 3$ case, and a glimpse at the $n > 3$ case. While our proofs do not explicitly depend on this section, it provides the key intuitions which motivate the succeeding sections.

4. Bijections between $\hat{\Sigma}(P)$and $Hull(E^P)$. As described in the previous section, the most natural way to prove our result would be to provide a bijection f obeying the syntactic correspondence. For $n = 2$ we can explicitly construct such a map. For higher dimensions we have no explicit map, although we are fairly certain

that the facts we develop later could, if one wanted, be used to prove existence of such a map. We emphasize that the proofs of our results do not rely on this section, and so we freely make claims here without proof.

The conditions we need on a bijection between $\hat{\Sigma}(P)$ and $Hull(E^P)$ are

(A) for $J \subseteq P$, each compressed schedule $\sigma \in \hat{\Sigma}(P)$ is mapped to the simplex $Hull(\{e^i : i \in J\})$;

(B) the image of any knowable set of $\hat{\Sigma}(P)$ is a relatively open subset of $Hull(E^P)$.

The first condition is fairly explicit; it says, for instance, that schedules of the form $(p)(p)(p)\ldots$ must be mapped to a corner vertex $\vec{e}^{\,p}$. The second relies on the notion of knowable sets, whose definition in terms of protocols is hard to deal with directly. We now state a combinatorial characterization of knowability. We don't prove this now, but we note that it is equivalent to the characterization proved below as Theorem A.4. If τ is a schedule fragment, the *cylinder* of τ, B_τ is the collection of all schedules that have τ as a prefix.

THEOREM 4.1. *A set S of compressed schedules is knowable if and only if the set $\{\sigma \in \Sigma(P) | \hat{\sigma} \in S\}$ can be expressed as a (possibly infinite) union of cylinders.*

Now suppose we can find a function f whose domain is the set $\Sigma(P)$ of all schedules (not just compressed ones), such that f satisfies the following four conditions:

(1) f maps $\Sigma(P)$ onto $Hull(E^P)$.

(2) Each schedule σ is mapped to the simplex $Hull(\{e^i : i \in J\})$, where $J = Active(\sigma)$.

(3) $f(\sigma) = f(\tau)$ if and only if σ and τ have the same compression.

(4) f maps schedules with a "large" common prefix to points that are "close." More precisely, there exists a function $\alpha(j)$ on the nonnegative integers that tends to 0 such that for any two schedules σ and ϕ, if σ and ϕ have a common prefix of j blocks, then $\|f(\sigma) - f(\phi)\| \leq \alpha(j)$, where $\|\cdot\|$ denotes the usual Euclidean length.

The second condition is just condition (A) above. It is not hard to show that conditions (1), (3), and (4) together with Theorem 4.1 imply condition (B). Note that given a map f satisfying (1) and (3), its restriction to the set $\hat{\sigma}$ of compressed schedules is a bijection.

The following definitions will be useful. A schedule σ is *degenerate* if it has a unique nonfaulty processor. Such a schedule is uniquely of the form $\tau[p]$ for some segment τ and processor p, where either τ is the null fragment or the last block B of τ is not equal to $\{p\}$. Writing τ as μB, we say that μ is the *fundamental fragment* of σ and B is the fundamental block. We also say that σ is p-degenerate. A degenerate schedule whose fundamental fragment has weight at most w is said to be *w-admissible*.

We now describe such a function f for the $n = 2$ case and give some indication how it can be extended to the $n \geq 3$ case.

4.1. A bijection for the 2-processor case. For simplicity, in the description of f, we consider the range to be the interval $[0, 1]$ instead of $Hull(E^{\{1,2\}})$.

Let us start by describing a mapping that does not work. Take each schedule σ and interpret it as an infinite ternary string $t = t(\sigma)$ by mapping $\{1\}$ to 0, $\{1, 2\}$ to 1, and $\{2\}$ to 2. Then any schedule can be interpreted as a real number between 0 and 1. Furthermore, this mapping is easily seen to satisfy properties (1), (2), and (4) above: the map is onto, it sends the schedule $[1] = \{1\}\{1\}\ldots$ to the endpoint 0 and $[2]$ to endpoint 1, and two schedules with a common prefix of j blocks map to points that differ by at most $(1/3)^j$. The problem is condition (3). The map sends the schedules $\{1\}[2]$ and $\{1, 2\}[1]$ to the point $1/3$, yet they do not have the same

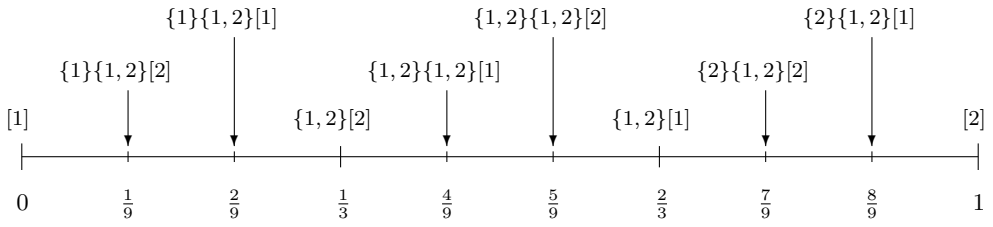

FIG. 1. *The map from* $\hat{\Sigma}(\{1,2\})$ *to the* $[0,1]$ *interval.*

compression, and $\{1\}[2]$ and $\{1,2\}[2]$ which have the same compression are mapped to $1/3$ and $2/3$, respectively. To fix this problem we modify $t = t(\sigma)$. Given $t = t(\sigma)$, define the infinite sequence a by $a_i = t_i$ if $t_1, t_2, \ldots, t_{i-1}$ has an even number of 1's and $a_i = 2 - t_i$ otherwise. Interpret a as a real number between 0 and 1 written in base 3. The map f is now defined to take $f(\sigma) = a$. It still satisfies (1), (2), and (4), but now it can also be shown to satisfy (3).

Figure 1 depicts the restriction of the mapping to compressed schedules. Under this map, degenerate schedules get mapped to rational numbers whose denominators are powers of 3. Observe that for $n = 2$, a compressed schedule σ is equal to the compression of some other schedule if and only if it is degenerate, in which case there is exactly one uncompressed schedule whose compression is σ. The mapping sends these two schedules to the same point. For example, the inverse image of $2/9$ is $\{1\}\{1,2\}[1]$ and $\{1\}\{2\}[1]$.

The bijection has a geometric description. Each prefix τ is associated with an interval R_τ which corresponds to the schedules that start with this prefix. The endpoints of the interval τ are the images of the schedules $\tau[1]$ and $\tau[2]$. The empty prefix corresponds to the entire interval, and if τ is a prefix of μ, then the interval corresponding to μ is a subinterval of that corresponding to τ. In general, the interval corresponding to τ is divided into three subintervals, corresponding to $\tau\{1\}, \tau\{1,2\}$, and $\tau\{2\}$.

It will be useful to describe this process of subdivision in *levels*. The level 0 subdivision is the subdivision into three intervals, corresponding to the prefixes $\{1\}, \{1,2\}$, and $\{2\}$. The level i subdivision is obtained from the level $i-1$ subdivision by taking each interval corresponding to a prefix of weight i (that is, the sum of the block sizes is i) and subdividing it into three subintervals as above. Thus the level 1 subdivision consists of the subdivision of the intervals $[0, 1/3]$ and $[2/3, 1]$ into three parts and the level 2 subdivision subdivides each of the intervals $[0, 1/9]$, $[2/9, 1/3]$, $[1/3, 2/3]$, $[2/3, 7/9]$, and $[8/9, 1]$ into three intervals.

4.2. The case of more than two processors. For $P = \{1,2\}$ we were able to give an explicit map from schedules to $Hull(E^P)$. For $|P| > 2$, we don't know how to do this. However, in the 2-processor case, we saw that the map can be defined by a process of successive subdivision. For each fragment τ we defined an interval R_τ which is the image of all schedules with prefix τ. For each schedule σ, the sequence of regions R_{σ^i}, where σ^i is the unique prefix of σ that appears in the level i subdivision, are nested, and their intersection is the image of σ. It should also be apparent that the lengths of the intervals into which we subdivided each interval are not critical; all we really needed was that the length of R_τ goes to 0 as we take larger and larger prefixes.

For $|P| \geq 3$ we will attempt to construct a map along similar lines. We will

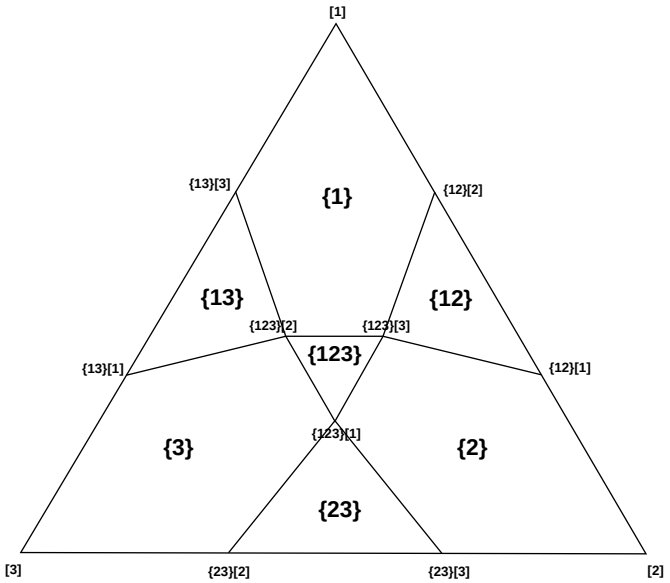

FIG. 2. *The subdivision* $D_0(\{1,2,3\})$.

construct a sequence $D_m(P)$ of decompositions of the simplex $Hull(E^P)$. The decomposition $D_0(P)$ consists of $2^{|P|}-1$ regions R_J, one for each nonempty subset of J of P; all schedules with first block J are mapped to R_J. This tiling is called the *level* 0 *decomposition* of the simplex. More generally, in the level m decomposition $D_m(P)$, $m \geq 0$, the regions correspond to fragments that are minimal subject to their weight being greater than m. The level $m+1$ decomposition is obtained from the level m decomposition by taking each region corresponding to a fragment τ of weight exactly m and tiling it by $2^n - 1$ regions corresponding to the fragments of the form τJ, where J is a block. The region R_τ is the image of the map applied to schedules with prefix τ. Given the level m decomposition for all m, the image of a schedule σ is determined as follows: for each m, σ is assigned to a unique $R_m(\sigma)$ region in the level m decomposition corresponding to the appropriate prefix of σ. The sequence of regions $R_m(\sigma)$ is nested and their diameters tend to 0, so σ is mapped to the unique point in their intersection. A key property will be that a point on the boundary of two or more regions will correspond to a set of schedules all having the same compression, where each schedule corresponds to one of the regions. The "vertices" of the decomposition $D_m(P)$ will correspond to schedules whose compression is an m-admissible degenerate schedule.

We now sketch how this can be done for $|P| = 3$; the same approach would also seem to work for higher dimensions. The combinatorial structure underlying the map is regular, but it is sufficiently complicated that writing a complete description and a rigorous proof that it works seems to be a very tedious undertaking, which is why we do not rely explicitly on this construction in the proof of our main result. The development in the later sections is closely related to the present construction; the reader will see that this construction is the basis for the "triangulation graphs" presented in section 6.2. Conversely, the interested reader can use the precise description of the triangulation graphs to get a precise description of the map in higher dimensions.

Let $P = \{1,2,3\}$. Figure 2 shows the level 0 decomposition, $D_0(P)$ into seven

regions. Each region is labeled by a nonempty subset of P. Each singleton set corresponds to a pentagonal region and each other set corresponds to a triangular region. All 0-admissible compressed degenerate schedules map to vertices in the figure. Each vertex is labeled by the unique compressed schedule that maps to it, and for each such vertex its inverse image is the set of all schedules whose compression is equal to its label. For example, the vertex labeled $\{1,2,3\}[2]$ lies on the boundary of the four regions $\{1\},\{3\},\{1,3\},\{1,2,3\}$ and for each such region there is a corresponding schedule that maps to that vertex: of $\{1\}\{2,3\}[2],\{3\}\{1,2\}[2],\{1,3\}[2]$, and $\{1,2,3\}[2]$. Consider a segment that separates two regions. Each point on the segment corresponds to two schedules, one that begins with the fragment defining the first region and one that begins with the fragment defining the other region. For example, the segment joining $\{1,2,3\}[2]$ and $\{1,3\}[1]$ separates the regions $\{3\}$ and $\{1,3\}$ and each point on the segment is the image of exactly two schedules $\{1,3\}\sigma$ and $\{3\}\{1\}\sigma$, where $\sigma \in \Sigma(\{1,2\})$.

In the higher level decompositions, we successively subdivide each region into seven subregions. When we focus on a region associated to fragment τ, it is useful to relabel each vertex of the region by the unique schedule beginning with τ that maps to that vertex. Thus, for instance, for the region $\{1,2\}$, we can view its vertices as $\{1,2\}[1]$, $\{1,2\}[2]$, and $\{1,2\}[3]$. When we subdivide this region we do it in a way analogous to the level 0 decomposition of the entire triangle, with $\{1,2\}[i]$ corresponding to the schedule $[i]$. Any triangular region is subdivided in this way.

To decompose a pentagonal region we treat it as a "distorted" triangle. In general, a pentagonal region will correspond to a fragment τ whose last block is a singleton set $\{i\}$. (The reader should follow this for the region labeled $\{1\}$.) The vertices of the region will correspond to the schedules $\tau[i]$, $\tau[j],\tau\{j,k\}[k],\tau\{j,k\}[j],\tau[k]$. When we subdivide, we view $\tau[i],\tau[j],\tau[k]$ as the vertices of the distorted triangle. The three segments joining $\tau[j]$ and $\tau[k]$ are together viewed as one "side" of the triangle. The two intermediate vertices $\tau\{j,k\}[k]$ and $\tau\{j,k\}[j]$ play the role of the vertices that are added when we subdivide that side.

Figure 3 depicts the level 1 decomposition with the vertices labeled and Figure 4 shows the decomposition with faces labeled. Note that the level 1 decomposition is produced from the level 0 decomposition by decomposing each of the three pentagons corresponding to regions whose fragment has weight 1.

The level 2 decomposition is not shown; it is obtained from the level 1 decomposition by subdividing each of the 12 regions that correspond to fragments of weight 2.

This completes our discussion of our attempt to prove Theorem 3.2 by an appropriate bijection between $Hull(E^P)$ and $\hat{\Sigma}(P)$. While we did not complete this approach, the constructions motivate much of what comes in the proof.

5. Reviewing the geometric case. As we stated, the proof of Theorem 3.2 is obtained by following closely the proof of its geometric analog Theorem 3.3. It is useful to review the outline of that proof.

Recall that we have a cover of the simplex $Hull(E^P)$ by relatively open sets $U_1,\ldots,U_n$ that satisfy the boundary property and we want to show that there is a point belonging to all of the sets. We begin with a lemma which applies to an arbitrary open cover of $Hull(E^P)$. For $\vec{x} \in Hull(E^P)$, let $B(x,\epsilon)$, the *closed ϵ-ball around x*, denote the set of points $\vec{y} \in Hull(E^P)$ within Euclidean distance ϵ of $\vec{x}$.

LEMMA 5.1. *For any finite collection $\mathcal{U}$ of open sets that covers $Hull(E^P)$ there is an $\epsilon = \epsilon(\mathcal{U}) > 0$ such that for each point $\vec{x} \in Hull(E^P)$, some member of $\mathcal{U}$ contains $B(\vec{x},\epsilon)$.*

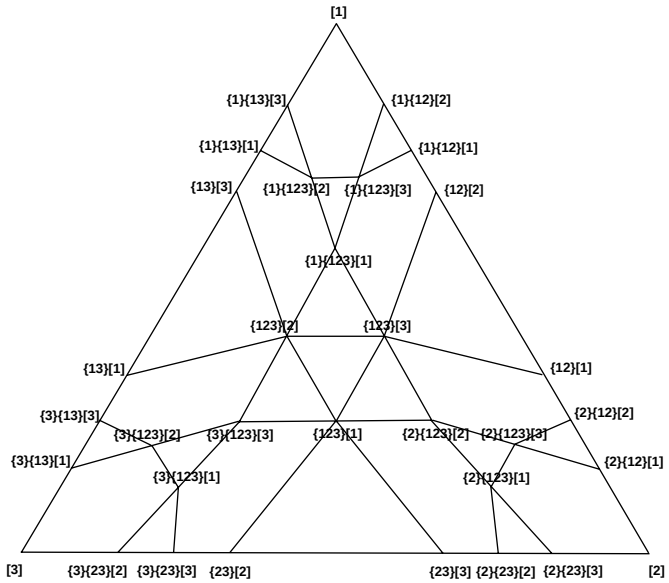

FIG. 3. *The subdivision* $D_1(\{1,2,3\})$ *with vertex labels.*

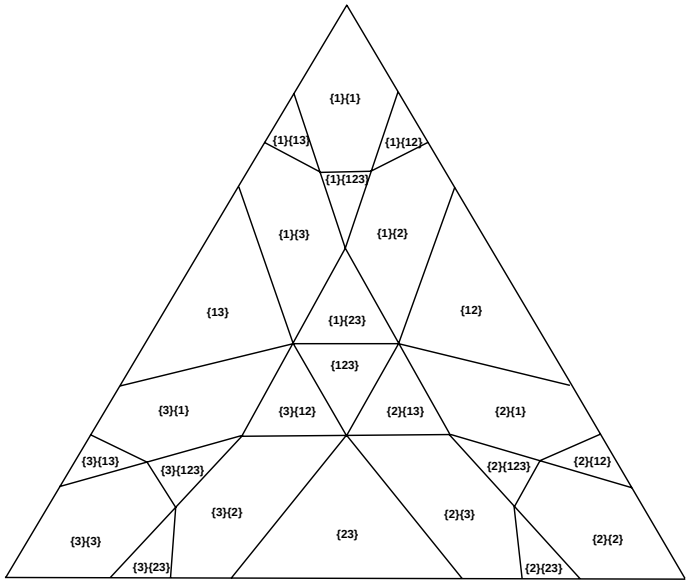

FIG. 4. *The subdivision* $D_1(\{1,2,3\})$ *with face labels.*

The key point is that the same ϵ works for all $\vec{x}$. Now given our covering $U_1, \ldots, U_n$ satisfying the boundary property we choose an $\epsilon > 0$ for which the conclusion of the above lemma is satisfied. We then define a function (labeling) λ of the points of $Hull(E^P)$ to $[n] = \{1, \ldots, n\}$ as follows: $\lambda(\vec{x})$ is the minimum j such that $B(\vec{x}, \epsilon) \subseteq U_j$. Note that, by the boundary property, this labeling satisfies the following *coherence condition*: for each $J \subset P$, each point of $Hull(E^J)$ is labeled by an element of J. It suffices to show that

(**) for some $\vec{y} \in Hull(E^P)$, $B(\vec{y}, \epsilon)$ contains n points with distinct labels, since then $\vec{y} \in \bigcap_{i=1}^{n} U_i$.

The proof of (**) relies on the notion of a *triangulation* of the simplex. Roughly, a triangulation T of $Hull(E^P)$ is a decomposition into a finite collection of n-vertex simplices with the property that any two simplices in the triangulation are disjoint or their intersection is a face of each (where by a face, we mean a subsimplex spanned by a subset of the vertices). The set of all simplices in T and faces of simplices in T are the *faces of* T.

For the ϵ given by Lemma 5.1, we construct a triangulation T satisfying the following.

LEMMA 5.2. *For each positive ϵ there exists a triangulation T of $Hull(E^P)$ all of whose simplices have diameter at most ϵ.*

Consider the labeling λ restricted to the vertices of T. The following lemma now completes the proof of (**) and Theorem 3.3.

LEMMA 5.3 (Sperner's lemma [1]). *Let $\lambda : Hull(E^P) \longrightarrow [n]$ be a coherent labeling and let T be a triangulation of $Hull(E^P)$. Then some n-vertex simplex of T has all of its vertices distinctly labeled.*

6. Proof of Theorem 3.2. Recall that we have knowable sets $K_1, K_2, \ldots, K_n$ that cover $\hat{\Sigma}(P)$ and satisfy the activity property. We wish to show that they have a nonempty intersection.

The first step in adapting the proof of Theorem 3.3 is to prove an analog of Lemma 5.1. Recall that for a fragment τ the set of schedules that are quasi extensions of τ is denoted Q_τ, and $\hat{Q}_\tau = \hat{\Sigma}P \cap Q_\tau$. The sets $\hat{Q}_\tau$ play the role of ϵ-balls. For example, for each compressed schedule ϕ, and for each integer w, let $\phi^{(w)}$ denote the maximal prefix of ϕ whose weight (sum of block sizes) is at most w. If we consider the sequence of sets $\hat{Q}_{\phi^{(w)}}$ we see that $\hat{Q}_{\phi^{(1)}} \supseteq \hat{Q}_{\phi^{(2)}} \supseteq \cdots$ and that the intersection of all of them is just ϕ itself. This is analogous to a sequence of balls of decreasing radius around a particular point. The analog of Lemma 5.1 is the following.

LEMMA 6.1. *Let $\mathcal{K}$ be a collection of knowable sets that covers $\hat{\Sigma}(P)$. Then there is an integer $w = w(\mathcal{K})$ with the property that for each schedule ϕ, some member of $\mathcal{K}$ contains $\hat{Q}_{\phi^{(w)}}$.*

Proof. Suppose for contradiction that for each w there is a schedule $\phi(w)$ such that the set $\hat{Q}_{\phi(w)^{(w)}}$ is not contained in any member of $\mathcal{K}$. Let $\Phi = \{\phi(w) : w \geq 1\}$. Construct a schedule σ as follows. Let σ_1 be any set that is the first block of infinitely many members of Φ, and inductively for $i > 1$, having defined $\sigma_1 \ldots \sigma_{i-1}$, let σ_i be a set such that $\sigma_1\sigma_2 \ldots \sigma_i$ is a prefix of infinitely many members of Φ. The compression $\hat{\sigma}$ must belong to some member $K \in \mathcal{K}$. By Proposition 2.25, there is a fragment τ of $\hat{\sigma}$ such that $\hat{Q}_\tau \subseteq K$. By construction of σ, τ is a prefix of infinitely many members of Φ. In particular, it belongs to some $\phi(w)$ with $w > w(\tau)$. Then $\hat{Q}_{\phi(w)^{(w)}} \subseteq \hat{Q}_\tau \subseteq K$ contradicts the choice of $\phi(w)$. □

Remark. The above result implies the fact mentioned in section 2.1 that the existence of a wait-free protocol Π for k-set agreement implies the existence of a bounded wait-free protocol. Given Π, define protocol Π', where each processor behaves as in Π except that if it sees that any processor has decided before it has decided, it immediately takes the lowest decision value it sees as its decision value. Clearly Π' is correct if Π is. Letting D_i be the set of schedules where some processor decides i, the above Lemma implies that there is a w such that after at most w total steps (by all of the processors) some processor has reached a decision. Thus, after taking at most $w + 2$ steps a processor will have decided and written its decision value.

Now given our covering $K_1, \ldots, K_n$ satisfying the activity property we choose an m for which the conclusion of the above lemma is satisfied. We then define a function (labeling) λ of the points of $\hat{\Sigma}(\Sigma^P)$ to $[n] = \{1, \ldots, n\}$ as follows: $\lambda(\hat{\sigma})$ is the minimum j such that $\hat{Q}_{\hat{\sigma}^{(m)}} \subseteq U_j$. Note that, by the activity property, this labeling satisfies the following *coherence condition*: for each $J \subset B$, each $\hat{\sigma} \in \hat{\Sigma}(\Sigma_J)$ is labeled by an element of J. It suffices to show the following analog of (**).

LEMMA 6.2. *Let $K_1, \ldots, K_n$ be knowable sets that cover $\hat{\Sigma}(P)$ and let λ be a labeling of the schedules that satisfies the coherence condition. Then for any integer $m \geq 1$, there is a fragment τ of weight at least m that is a prefix of n schedules having different labels.*

The proof of this relies on abstracting the notion of a *triangulation* for the set of compressed schedules and proving an analog of Sperner's lemma.

6.1. Triangulation graphs. By analyzing the proof of Sperner's lemma, it can be seen that the lemma can be interpreted as a statement about graphs embedded in $Hull(E^P)$ that have certain properties. This motivates the definition of the following class of graphs defined on $\hat{\Sigma}(P)$.

A *triangulation graph* on $\hat{\Sigma}(P)$ is a finite graph $G = (V, E)$ whose vertex set is a subset of $\hat{\Sigma}(P)$ and which satisfies the following properties:

(1) For each $p \in P$, the schedule $[p] = \{p\}\{p\}\{p\}\ldots$ is a vertex.
(2) If C is a clique contained in $\hat{\Sigma}(J)$ of size $|J| - 1$, then
   (a) if C is contained in $\hat{\Sigma}(I)$ for any proper subset I of J, then there is a unique clique of size $|J|$ in $\hat{\Sigma}(J)$ that contains C;
   (b) if C is not contained in $\hat{\Sigma}(I)$ for any proper subset I of J, then there are exactly two cliques of size $|J|$ in $\hat{\Sigma}(J)$ that contain C.

Here we use *clique* to mean a not necessarily maximal complete subgraph.

These conditions correspond to those satisfied by the skeleton of a triangulation in the geometric case. The first condition comes by associating the schedules $[p]$ to the generators $\vec{e}^p$ of the simplex $Hull(E^P)$, which are necessarily vertices of any triangulation. The second condition corresponds to the fact that in any triangulation of the simplex, if a $(|J|-1)$-vertex face of T lies in $Hull(E^J)$, then (i) if it lies on the boundary of $Hull(E^J)$, then it is a face of a unique $|J|$-vertex face of T, while (ii) if it lies in the interior of $Hull(E^J)$, then it is the common boundary of a pair of $|J|$-vertex faces of T.

These conditions are sufficient to prove an analog of Sperner's lemma.

LEMMA 6.3. *Let $\lambda : \hat{\Sigma}(P) \longrightarrow [n]$ be a coherent labeling and let G be a triangulation graph on $\hat{\Sigma}([n])$. Then some n-vertex clique of G has all of its vertices labeled differently.*

Proof. For $k \in [n]$, let $g(k)$ be the number of k vertex cliques in $\hat{\Sigma}([k])$ whose vertices are labeled differently. Since λ is coherent these labels must be $\{1, 2, \ldots, k\}$. The key step is the following claim.

Claim. For each k between 2 and n, $g(k) \equiv g(k-1) \mod 2$.

It then follows that $g(n) \equiv g(1) \equiv 1 \mod 2$ since the schedule [1] is the unique vertex in $\hat{\Sigma}([1])$, and we conclude that $g(n) \neq 0$.

It suffices to prove the claim. For k between 2 and n, define $p(k)$ to be the number of pairs (C', C), where C is a k-clique in $\hat{\Sigma}([k])$ and C' is a $(k-1)$-clique contained

in C that is distinctly labeled $1, 2, \ldots, k-1$. Then the claim follows from

$$g(k) \equiv p(k) \mod 2,$$

$$p(k) \equiv g(k-1) \mod 2.$$

The first relation is obtained by computing $p(k)$ in the following manner. Each k-clique C in $\hat{\Sigma}([k])$ which is not distinctly labeled contains either 0 or 2 cliques of size $k-1$ that are labeled $1, 2, \ldots, k-1$ and so contribute 0 mod (2) to $p(k)$. On the other hand, each distinctly labeled k-clique C has a unique subclique of size $k-1$ that is labeled $1, 2, \ldots, k-1$ and so contributes 1 to $p(k)$.

The second relation follows by considering, for each $(k-1)$-clique C' with labels $1, 2, \ldots, k-1$, the number of k-cliques of $\hat{\Sigma}([k])$ to which it belongs. By the definition of triangulation graph, if C' does not lie in $\hat{\Sigma}(I)$ for some proper subset I of $[k]$, then it belongs to exactly two cliques C of size k in $\hat{\Sigma}([k])$ and so contributes 0 mod (2) to $p(k)$. On the other hand, if $C' \subset \hat{\Sigma}(I)$ for some proper subset I of $[k]$, then the definition of triangulation graph implies that C' is in a unique k-vertex clique $C \subset \hat{\Sigma}([k])$ and so contributes exactly 1 to $p(k)$. Thus $p(k) \mod 2$ counts the parity of the number of $(k-1)$-cliques C' labeled by $1, 2, \ldots, k-1$ that are in $\hat{\Sigma}(I)$ for some I properly contained in $[k]$. But the fact that C' contains all labels $1, 2, \ldots, k-1$ implies, by the coherence of λ, that I contains $[k-1]$ and so C' must be in $\hat{\Sigma}([k-1])$. Therefore, $p(k) \equiv g(k-1) \mod 2$. □

Next we will prove the following lemma.

LEMMA 6.4. *For any integer w, there exists a triangulation graph G on $\hat{\Sigma}(P)$ with the property that for any clique of G there is a fragment τ of weight at least w such that every vertex of the clique is contained in $\hat{Q}_\tau$.*

Together with Lemma 6.3, this lemma completes the proof of Lemma 6.2 and hence of Theorems 3.2 and 1.1.

Finally, it remains to show the existence of the desired triangulation graphs. This turns out to be the most technically arduous part of the proof.

6.2. Constructing triangulation graphs. We will define a sequence of triangulation graphs $\{G_m(P) | m \geq 0\}$ on $\hat{\Sigma}(P)$ with the property that any clique of $G_m(P)$ is a subset of $\hat{Q}_\tau$ for some fragment τ of weight at least $w = m - n$ (where $n = |P|$).

The precise description of $G_m(P)$ is technical, but the "picture" of the construction is nice. The graph $G_m(P)$ is closely related to the decomposition $D_m(P)$, described in section 4, but since we have only given a hint as to the general construction of $D_m(P)$, we cannot use the $D_m(P)$ explicitly in our formal description of the $G_m(P)$. Nevertheless to help understand the technical description, the reader should compare the graphs $G_0(P)$ and $G_1(P)$ depicted in Figures 5 and 6 with the pictures of $D_0(P)$ and $D_1(P)$ in section 4. The following properties are evident from the examples and will also hold in general:

(1) The vertex set of $G_m(P)$ corresponds to the vertex set of $D_m(P)$, i.e., it is the set $V_m(P)$ of m-admissible compressed degenerate schedules in $\Sigma(P)$.
(2) The graph $G_m(P)$ is obtained from the decomposition $D_m(P)$ by adding edges in order to triangulate the pentagonal regions in $D_m(P)$. Recall that each pentagonal face is labeled by a fragment that ends with a singleton block, i.e., one of the form $\tau = \mu\{i\}$. The pentagon is triangulated by connecting the vertex corresponding to (the compression of) $\tau[i]$ to the vertices corresponding to (the compression of) $\tau\{j,k\}[j]$ and $\tau\{j,k\}[k]$.

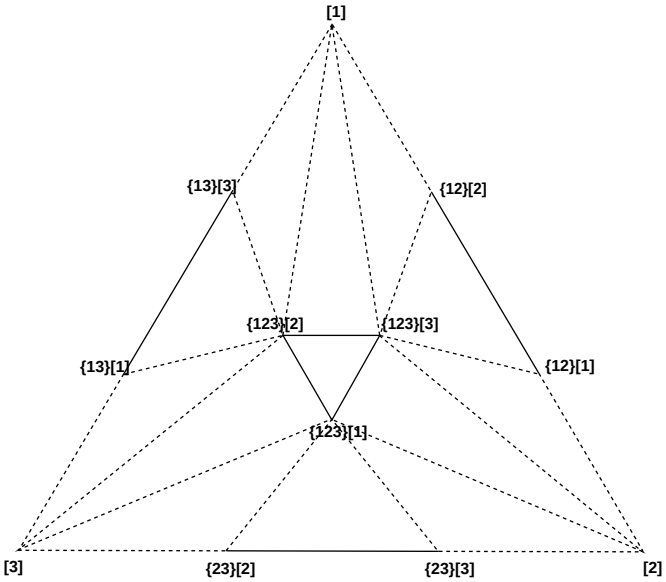

FIG. 5. *The graph* $G_0(\{1,2,3\})$.

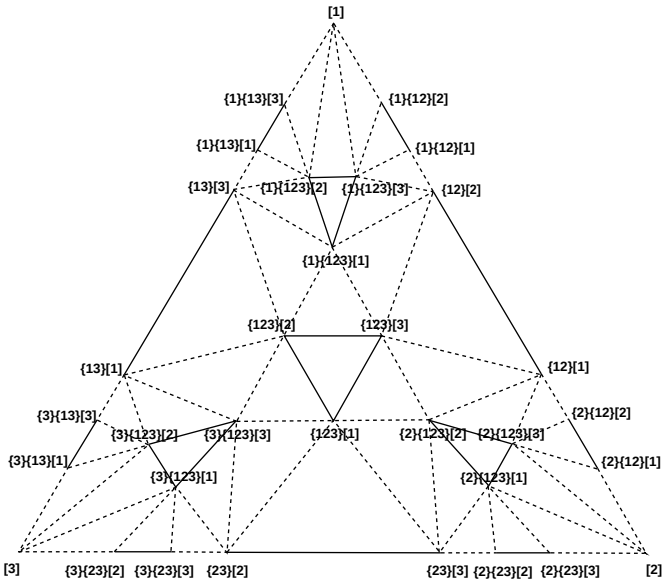

FIG. 6. *The graph* $G_1(\{1,2,3\})$.

(3) The edges in $G_m(P)$ are depicted as either solid or dashed lines. The reason for this distinction will be clear only after we give a precise definition of the graphs.

Let us now give the formal definition of the graphs $G_m(P)$. A schedule σ is *degenerate* if it has a unique nonfaulty processor. Let $V(J)$ denote the set of compressed degenerate schedules σ with $Active(\sigma) \subseteq J$. We say that $\sigma \in V(J)$ is p-degenerate if p is the unique nonfaulty processor, and we denote by $V^p(J)$ the set of p-degenerate

schedules. Note that a p-degenerate schedule can be written in the form $\tau[p]$, where τ is a fragment and $[p]$ denotes the schedule all of whose blocks are singleton p blocks.

Recall that the weight of a fragment is the sum of its block sizes. For an arbitrary schedule σ and nonnegative integer m, let $\sigma^{(m)}$ be the maximal prefix of σ having weight less than or equal to m and let $B_m(\sigma)$ be the block of σ following $\sigma^{(m)}$. We say that σ is *m-admissible* if (i) it belongs to $V(P)$ and (ii) all blocks after $B_m(\sigma)$ are singleton p blocks. (This definition is equivalent to the definition of m-admissible given in section 4.) Thus an m-admissible schedule is of the form $\sigma^{(m)}B_m(\sigma)[p]$. Note that if σ is m-admissible, then it is j-admissible for all $j \geq m$. Also, observe that since σ is compressed, $p \in B_m(\sigma)$. We denote by $V_m^p(J)$ the set of m-admissible p-degenerate schedules in $V(J)$ and $V_m(J)$ is the union over p of $V_m^p(J)$. $V_m(P)$ is the vertex set of $G_m(P)$. Note that for $j \geq m$, this implies that the vertex set of $G_m(P)$ is a subset of the vertex set of $G_j(P)$.

We now define the edge set of $G_m(P)$. The edge set is the union of two relations: m-similarity, which is an equivalence relation, and m-shadowing, which is acyclic.

Two m-admissible schedules σ and ϕ in $V_m(P)$ are *m-similar* if $\sigma^{(m)} = \phi^{(m)}$ and $B_m(\phi) = B_m(\sigma)$. This is clearly an equivalence relation.

If $\sigma \in V_m^p(P)$ and $\phi \in V_m^q(P)$, we say that σ *m-shadows* ϕ if there is a set T satisfying $q \in T \subseteq B_m(\sigma) - \{p\}$ such that the compression of the fragment $\sigma^{(m)}T$ is equal to $\phi^{(m)}B_m(\phi)$. In particular this implies that ϕ is equal to the compression of $\sigma^{(m)}T[q]$. The m-shadow relation is acyclic since σ m-shadows ϕ implies that $w(\sigma^{(m)}B_m(\sigma)) > w(\phi^{(m)}B_m(\phi))$.

EXAMPLE 6.1. *Consider the compressed schedules:*

$$\begin{aligned}
\sigma &= \{1,2\}\{2,3\}\{1,2\}[2],\\
\rho &= \{1,2\}\{1,2,3\}[1],\\
\phi &= \{1,2\}\{1,2,3\}[2],\\
\nu &= \{1,2\}\{1,3\}[3],\\
\mu &= \{1,2\}[1].
\end{aligned}$$

Each of the schedules is in $V_m(P)$ for $m \geq 4$, all but σ also belong to $V_2(P)$ and $V_3(P)$, and μ belongs to $V_0(P)$ and $V_1(P)$. σ 4-shadows ρ, and is otherwise unrelated to all other schedules. ρ is m-similar to ϕ for $2 \leq m \leq 4$ and does not m-shadow any of the other schedules for any m. ϕ 2-shadows both μ and ν and 3-shadows ν. ν 2-shadows μ.

We can now define the edge set $E_m(P)$ of the graph $G_m(P)$. The pair $(\sigma, \phi) \in E_m(P)$ if either (i) σ and ϕ are m-similar, (ii) σ m-shadows ϕ, or (iii) ϕ m-shadows σ. In Figures 5 and 6, the solid edges correspond to those that come from m-similarity and the dashed edges arise from m-shadowing. It should be emphasized that while $V_i(P) \subset V_j(P)$ for $i < j$, the edge sets are not nested.

To complete the proof of Lemma 6.4 and hence the proof of the impossibility of wait-free m-set agreement it is now enough to prove two facts, as follows.

LEMMA 6.5.

(1) *The graph G_m is a triangulation graph for $\hat{\Sigma}(P)$.*
(2) *For any clique C in $G_m(P)$, C is contained in a set of the form $\hat{Q}_\tau$ for some fragment τ with $m - n < w(\tau)$.*

$G_m(P)$ trivially satisfies the first condition of triangulation graphs since $[p]$ is m-admissible for all $m \geq 0$.

TABLE 2
Summary of terminology for cliques and flags.

$\sigma^{(m)}$	the largest prefix of σ with weight $\leq m$
$B_m(\sigma)$	the first block of σ following $\sigma^{(m)}$
σ is m-admissible	σ is compressed and of the form $\sigma^{(m)}B_m(\sigma)[p]$
flag $\mathcal{F}$	a finite family of nested sets including $\emptyset$
$\mathcal{F}$ is a J-flag	the largest set in $\mathcal{F}$ is J
$\mathcal{F}^+(p)$	the unique smallest set in $\mathcal{F}$ containing element p
I-clique	consists of a p-degenerate vertex for each $p \in I$
(I, J)-clique	I-clique contained in $V_m(J)$
$(\tau, \mathcal{F})$ is (m, J)-admissible	τ a fragment, $\mathcal{F}$ a J-flag, $Active(\tau) \subseteq J$, $w(\tau) \leq m$, and $w(\tau F) > m$ for $F \in \mathcal{F} - \{\emptyset\}$
$C_I(\tau, \mathcal{F})$	$\{\sigma^p \mid p \in I\}$ where σ^p is the compression of $\tau\mathcal{F}^+(p)[p]$
$(\tau, \mathcal{F})$ is a (m, J)-flag representation of I-clique C	$C = C_I(\tau, \mathcal{F})$ and $(\tau, \mathcal{F})$ is (m, J)-admissible

The hard part is proving that $G_m(P)$ satisfies the second condition of triangulation graphs. The key is to obtain a complete characterization of the cliques of $G_m(J)$. Along the way we will also prove the second part of Lemma 6.5 (which will follow from Lemma 6.7). We advise the reader to refer frequently to Figure 6 to help in understanding what follows.

Alas, we need some more definitions. (Table 2 provides a summary of some of the key definitions.) Let $J \subseteq P$. A *flag* is a finite family $\mathcal{F}$ of sets that are totally ordered by inclusion and includes the emptyset. The unique maximal set of $\mathcal{F}$ is denoted $\mathcal{F}^M$. If $J = \mathcal{F}^M$ we say that $\mathcal{F}$ is a J-flag. For $p \in J$, $\mathcal{F}^+(p)$ denotes the unique smallest set containing p, and $\mathcal{F}^-(p)$ denotes the unique largest set not containing p.

By the definition of the edge set, any clique C consists of schedules that are p-degenerate for distinct p. If $I \subseteq P$ and C is a clique that consists of one p-degenerate schedule for each $p \in I$, then C is a called an *I-clique*. We typically denote such a clique by $\{\sigma(p) | p \in I\}$, where $\sigma(p)$ denotes a p-degenerate schedule.[1] As noted when defining m-admissibility, each $\sigma(p)$ is equal to $\sigma(p)^{(m)}B_m(\sigma(p))[p]$. If C is an I-clique all of whose vertices belong to $V_m(J)$, we say that C is an (I, J)-clique.

Let $C = \{\sigma(p) | p \in I\}$ be an (I, J)-clique. A processor $q \in I$ is said to be *dominant* in C if it maximizes $w(\sigma(p)^{(m)}B_m(\sigma(p)))$ among all $p \in I$. The following lemma provides a simple combinatorial representation for (I, J)-cliques.

LEMMA 6.6. *Let $C = \{\sigma(p) | p \in I\}$ be an I-clique in $G_m(J)$. Let q be a dominant processor and $\tau = \sigma(q)^{(m)}$. Then*

(1) *for each $p \in I$, there is a subset F_p of J containing p such that $\sigma(p)$ is equal to the compression of $\tau F_p[p]$;*

[1] A remark on notation: The (p) in $\sigma(p)$ is simply an index, and so $\sigma(p)$ in this case stands for a p-degenerate schedule. This should not be confused with the notation $\sigma[p]$, which denotes a schedule consisting of a *fragment* σ followed by an infinite sequence of $\{p\}$ blocks.

(2) *the family of sets* $\mathcal{F} = \{F_p | p \in I\} \cup \{\emptyset, J\}$ *is a* J*-flag;*
(3) *for each* $p \in I$, $F_p = \mathcal{F}^+(p)$, *and* $w(\tau F_p) > m$.

Proof. Since q is dominant in C, then for each $p \in I - \{q\}$, $\sigma(q)$ either is m-similar to or m-shadows $\sigma(p)$, which implies that for $p \neq q$, $\sigma(p)$ is the compression of a schedule of the form $\tau F_p[p]$, where $p \in F_p \subset J$. This proves the first part. For the second part, let $p, r \in I$. If $\sigma(p)$ is m-similar to $\sigma(r)$, then the condition that the compression of τF_p is equal to the compression of τF_r implies $F_p = F_r$. Otherwise $\sigma(p)$ m-shadows $\sigma(r)$, which means that $\sigma(r)^{(m)} B_m(\sigma(r))$ can be written as the compression of $\sigma(p)^{(m)} B$ for some $B \subseteq B_m(\sigma(p)) - \{p\}$. Since τF_r and $\sigma(p)^{(m)} B$ have the same compression and τF_p and $\sigma(p)^{(m)} B_m(\sigma(p))$ have the same compression (by comparing the total number of steps taken by each processor in each of these fragments), we conclude that $F_r \subseteq F_p - \{p\}$. From this it follows that $\mathcal{F} = \{F_p | p \in I\} \cup \{\emptyset, J\}$ is a J-flag and for each $p \in I$, $F_p = \mathcal{F}^+(p)$, completing the second part. Finally, the m-admissibility of the vertices in C implies that $w(\tau F) \geq m$ for all nonempty $F \in \mathcal{F}^+(p)$. □

This motivates the following definitions. An (m, J)-*admissible pair* is a pair $(\tau, \mathcal{F})$, where $\tau \in \Phi(J)$ (recall that $\Phi(J)$ is the set of fragments whose blocks are subsets of J) and $\mathcal{F}$ is a J-flag such that $w(\tau) \leq m$ and $w(\tau F) > m$ for all nonempty $F \in \mathcal{F}$. For such a pair, we define $C_I(\tau, \mathcal{F})$ to be the set $\{\sigma(p) | p \in I\}$ of schedules where $\sigma(p)$ is the compression of the schedule $\tau \mathcal{F}^+(p)[p]$. Note that since $p \in \mathcal{F}^+(p)$ this is equal to the schedule obtained by compressing the fragment $\tau \mathcal{F}^+(p)$ and appending $[p]$.

From Lemma 6.6 we have that every (I, J)-clique is of the form $C_I(\tau, \mathcal{F})$ for some (m, J)-admissible pair. In fact, the converse of this statement also holds and we state them together in the following lemma.

LEMMA 6.7.
(1) *For* $I \subseteq J \subseteq P$, *each* (I, J) *clique of* $G_m(P)$ *has the form* $C_I(\tau, \mathcal{F})$ *for some* (m, J) *admissible pair* $(\tau, \mathcal{F})$.
(2) *If* $(\tau, \mathcal{F})$ *is an* (m, J)*-admissible pair and* $I \subseteq J$, *then* $C_I(\tau, \mathcal{F})$ *is an* (I, J)*-clique.*

Proof. The first part follows from Lemma 6.6. To prove the second part, suppose that $(\tau, \mathcal{F})$ is m-admissible. We first must show that each schedule in $C_I(\tau, \mathcal{F}) = \{\sigma(p) | p \in I\}$ is a vertex of $V_m(J)$. Proposition 6.8 follows easily from the definition of compression.

PROPOSITION 6.8. *If* $(\tau, \mathcal{F})$ *is* (m, J) *admissible,* $F = \mathcal{F}^+(p)$, *and* $p \in J$, *then the compression of* $\tau F[p]$ *can be written in the form* $\mu B[p]$, *where* $w(\mu) \leq w(\tau)$ *and* $w(\mu B) = w(\tau F)$. *Thus* $\sigma(p)$ *is* m*-admissible and so belongs to* $V_m(J)$.

Next we must show that for $p, q \in I$, $\sigma(p)$ and $\sigma(q)$ are adjacent in $G_m(P)$, i.e., either they are m-similar or one m-shadows the other. Let $A = \mathcal{F}^+(p)$ and $B = \mathcal{F}^+(q)$. Thus $\sigma(p)$ is the compression of $\tau A[p]$ and $\sigma(q)$ is the compression of $\tau B[q]$, and also $w(\tau A)$ and $w(\tau B)$ are both greater than m. If $A = B$, then $\tau A[p]$ and $\tau A[q]$ have exactly the same hidden blocks, and so $\sigma(p)$ and $\sigma(q)$ are clearly m-similar. If $A \neq B$, then without loss of generality $A \subset B$. Furthermore, $p \in A$ and $q \in B - A$, since $A = \mathcal{F}^+(p)$ and $B = \mathcal{F}^+(q)$. Let $\sigma(q) = \mu B'[q]$ be the compression of $\tau B[q]$. Then $B' = B \cup C$, where C is the union of some number of blocks (possibly 0) at the end of τ. Also, $w(\mu B') = w(\tau B)$. Every hidden block of $\tau B[q]$ is also hidden in $\tau A[p]$, and so $\tau A[q]$ can be partially compressed to $\mu A'[p]$, where $A' = A \cup C$. This implies that $p \in A' \subseteq B' - \{q\}$, and so $\sigma(q)$ m-shadows $\sigma(p)$. □

This lemma provides a nice combinatorial characterization of cliques. If C is an I-clique and $C = C_I(\tau, \mathcal{F})$, where $(\tau, \mathcal{F})$ is an (m, J)-admissible pair, then $(\tau, \mathcal{F})$

is called an (m, J)-*flag representation* of C. This representation is in general not unique. Lemma 6.6 gave such a representation for each I-clique C in $V_m(J)$. This construction has the properties that τ is equal to the prefix $\sigma(q)^{(m)}$ for some (in fact, any) dominant processor q and that $\mathcal{F}$ consists exactly of those sets F_p for $p \in I$, where F_p is the unique set such that $\sigma(p)$ is the compression of $\tau F_p[p]$. In particular, $\mathcal{F}$ is the unique minimal J-flag for which $(\tau, \mathcal{F})$ is a (k, J)-flag representation of C. We call this representation the (m, J)-*canonical* representation of C. It is clear that the (m, J)-canonical representation of the I-clique C is unique.

EXAMPLE 6.2. *Let* $m = 2$ *and* $P = \{1, 2, 3, 4, 5\}$. *The set consisting of* $\{1, 2, 3\}[3]$, $\{1, 2\}\{1, 3\}[1]$, *and* $\{1, 2\}\{1, 5, 3, 4\}[4]$ *has four* $(2, P)$*-flag representations:* $(\tau, \mathcal{F})$, $(\mu, \mathcal{F})$, $(\tau, \mathcal{G})$, *and* $(\mu, \mathcal{G})$, *where* $\mathcal{F} = \{\emptyset, \{3\}, \{1, 3\}, \{1, 3, 4, 5\}, \{1, 2, 3, 4, 5\}\}$, $\mathcal{G} = \{\emptyset, \{3\}, \{1, 3\}, \{1, 3, 5\}, \{1, 3, 4, 5\}, \{1, 2, 3, 4, 5\}\}$, $\tau = \{1, 2\}$, *and* $\mu = \{2\}\{1\}$. *The canonical representation is* $(\tau, \mathcal{F})$.

We can now prove the second part of Lemma 6.5. If C is any I-clique, let $(\tau, \mathcal{F})$ be an (m, P)-flag representation of C. The (m, P)-admissibility of $(\tau, \mathcal{F})$ implies that $w(\tau) > m - n$. Every schedule in C is the compression of a schedule of the form $\tau F[p]$ for some $F \in \mathcal{F}$ and $p \in I$ and is thus a quasi extension of τ. Thus $\sigma \in \hat{Q}_\tau$ as required.

It remains only to check that $G_m(P)$ satisfies the third condition of the definition of triangulation graphs. We will prove the following.

LEMMA 6.9. *Let* $I \subseteq J \subseteq P$ *and let* C *be an* I*-clique in* $G_m(J)$. *Then the number of* (J, J)*-cliques that contain* C *is equal to the number of distinct* J*-flag representations of* C.

In light of this lemma, the two parts of the second property of triangulation graphs follow, respectively, from the two parts of the following lemma.

LEMMA 6.10. *Let* $J \subseteq P$, $p \in J$, *and* $I = J - \{p\}$. *Let* C *be an* I*-clique that is contained in* $V_m(J)$.

(1) *If* C *is contained in* $V_m(I)$, *then* C *has a unique* J*-flag representation.*

(2) *If* C *is not contained in* $V_m(I)$, *then* C *has exactly two* J*-flag representations.*

Thus, all that remains is to prove Lemmas 6.9 and 6.10, which we now do. We first make some preliminary observations about flags and flag representations.

PROPOSITION 6.11. *Let* $\mathcal{F}$ *and* $\mathcal{H}$ *be* J*-flags. If* $\mathcal{H}^+(r) = \mathcal{F}^+(r)$ *for every* $r \in J$, *then* $\mathcal{H} = \mathcal{F}$.

Proof. Suppose that $\mathcal{H} \neq \mathcal{F}$ are J-flags and let A be a set that is in one but not the other, say, it is in $\mathcal{H}$ but not in $\mathcal{F}$. Then there is an element r such that $\mathcal{H}^+(r) = A$, and so $\mathcal{H}^+(r) \neq \mathcal{F}^+(r)$. □

LEMMA 6.12. *Let* $C = \{\sigma(p) | p \in I\}$ *be an* (I, J)*-clique and let* q *be a dominant processor. Let* $(\tau, \mathcal{F})$ *be the canonical* (m, J)*-flag representation of* C. *Let* $(\mu, \mathcal{H})$ *be an arbitrary* (m, J)*-flag representation of* C. *Then*

(1) $\sigma(q) = \tau\mathcal{F}^+(q)[q]$.

(2) μ *can be written in the form* $\nu\lambda$, *where* ν *is a fragment such that the compression of* $\nu\mathcal{F}^+(q)$ *is* $\tau\mathcal{F}^+(q)$ *and* λ *is a possibly empty fragment consisting of pairwise disjoint blocks.*

(3) *Let* B *denote the union of the blocks of* λ. *Then* $B \subseteq J - I$ *and* $\mathcal{H}^+(p) = \mathcal{F}^+(p) - B$ *for each* $p \in I$.

(4) $I \subseteq \mathcal{H}^+(q) \subseteq \mathcal{F}^+(q) \subseteq J$.

Proof. The first part follows immediately from the definition of the canonical (m, J)-flag representation. For the second part, note that $\mu\mathcal{H}^+(q)[q]$ must compress to $\sigma(q) = \tau\mathcal{F}^+(q)[q]$, which means that $\mu\mathcal{H}^+(q)$ must compress to $\tau\mathcal{F}^+(q)$. Let ν be

the portion of μ that compresses to τ and let λ be the portion of μ that is merged with $\mathcal{H}^+(q)$ to form $\mathcal{F}^+(q)$. Then $\mu = \nu\lambda$, and λ must consist of disjoint blocks. For the third part, if $p \in I$, then $\tau\mathcal{F}^+(p)$ must have the same compression as $\nu\lambda\mathcal{H}^+(p)$, which means that $\lambda\mathcal{H}^+(p)$ must compress to $\mathcal{F}^+(p)$, so $\mathcal{H}^+(p) = \mathcal{F}^+(p) - B$. Since $p \in \mathcal{H}^+(p)$, we must have $p \notin B$, so $B \subset J - I$. For the fourth part, $\mathcal{H}^+(q)$ contains $\mathcal{H}^+(p)$ for all $p \in I$, so $I \subseteq \mathcal{H}^+(q)$. □

LEMMA 6.13. *Each (J, J)-clique has a unique (m, J)-flag representation.*

Proof. Let $C = \{\sigma(p) : p \in J\}$ be a J-clique contained in $V_m(J)$ and let $(\tau, \mathcal{F})$ be its J-canonical representation. Suppose that $(\mu, \mathcal{H})$ is any other J-flag representation. Since $\mathcal{H}$ is a J-flag, there must be $q \in J$ such that $\mathcal{H}^+(q) = J$. Then $\mu J[q]$ is compressed and must be equal to $\sigma(q)$. Furthermore q is dominant in C, so by the definition of the canonical representation $\tau = \mu$.

By Proposition 6.11, if $\mathcal{G} \neq \mathcal{F}$, then there exists r such that $\mathcal{F}^+(r) \neq \mathcal{G}^+(r)$. But then the compression of $\tau\mathcal{F}^+(r)[r]$ cannot be equal to the compression of $\tau\mathcal{G}^+(r)[r]$. Therefore $\mathcal{F} = \mathcal{G}$ and the (m, J)-flag representation is unique. □

Now we are ready to prove Lemmas 6.9 and 6.10, to finish the proof of the main theorem.

Proof of Lemma 6.9. Let C be an (I, J)-clique, and let $C^1, C^2, \ldots, C^r$ be the distinct (J, J)-cliques that contain C. By Lemma 6.13, each of the C^i has a unique (m, J)-flag representation, $(\tau_i, \mathcal{F}_i)$, and by the definition of the representation, $(\tau_i, \mathcal{F}_i)$ is also an (m, J)-flag representation of C, i.e., $C = C_I(\tau_i, \mathcal{F}_i)$. Also, these are the only (m, J)-flag representations of C, since any such representation for C is also an (m, J)-flag representation of some J-clique containing C. □

Proof of Lemma 6.10. Let $C = \{\sigma(r) | r \in I\}$ be an $I = J - \{p\}$-clique and let $(\mu, \mathcal{H})$ denote an arbitrary (m, J)-flag representation of C. Let q denote a dominant processor of C. From Lemma 6.12, $I \subseteq \mathcal{H}^+(q) \subseteq J$.

To prove the first part of the lemma, suppose that C is contained in $V_m(I)$. Then $\mathcal{H}^+(q) \neq J$ so $\mathcal{H}^+(q) = I$. Let $\mathcal{G} = \mathcal{H} - \{J\}$. Then $(\mu, \mathcal{G})$ is a (m, I)-flag representation of C. But, by Lemma 6.13, there is only one such representation, so this implies that $(\mu, \mathcal{H})$ must be the unique (m, J)-flag representation.

We proceed to the second part of the lemma. Let $(\tau, \mathcal{F})$ be the canonical (m, J)-flag representation of C and let q be a dominant processor of C. We want to show that there is exactly one other representation. Now, by Lemma 6.12, either $\mathcal{F}^+(q) = J$ or $\mathcal{F}^+(q) = I$. We proceed by analyzing these two cases separately.

Case I. $\mathcal{F}^+(q) = I$. First we construct another (m, J)-flag representation. Let S be the last block of τ that contains p. There is such a block since, by hypothesis, $C \not\subset V_m(J - \{p\})$. If $S = \{p\}$, then $\tau\mathcal{F}^+(q)$ would not be compressed, contradicting the definition of the canonical representation. So $S \neq \{p\}$. Let ψ be the sequence obtained by replacing the block S by $\{p\}$ followed by $S - \{p\}$. Then $(\psi, \mathcal{F})$ is also an (m, J)-representation of C.

Now we show that this is the only other (m, J)-flag representation of C. Let $(\mu, \mathcal{H})$ be an arbitrary (m, J)-flag representation of C. Then, by Lemma 6.12, $I \subseteq \mathcal{H}^+(q) \subseteq \mathcal{F}^+(q)$ implies that $\mathcal{H}^+(q) = \mathcal{F}^+(q) = I$. Defining ν, λ, B as in Lemma 6.12 we must have $B = \emptyset$ and λ is the empty string. Then $\mathcal{H}^+(r) = \mathcal{F}^+(r)$ for all $r \in I$, and also $\mathcal{H}^+(p) = \mathcal{F}^+(p) = J$, so Proposition 6.11 implies $\mathcal{H} = \mathcal{F}$. Finally, $\mu I[q]$ must compress to $\tau I[q]$. Then either $\mu I[q]$ is already compressed (and $\mu = \tau$) or μ contains a hidden block. But a block in $\mu I[q]$ can be hidden only if it is disjoint from I, i.e., it is a singleton p block, and it must be the last appearance of p. This means that μ is equal to ψ above and so $(\mu, \mathcal{H}) = (\psi, \mathcal{F})$.

Case II. $\mathcal{F}^+(q) = J$. Then $J - \{p\}$ is not in $\mathcal{F}$. Again, we must construct another (m, J)-flag representation of C and show that this is the only other one. First note the following.

PROPOSITION 6.14. *If $(\mu, \mathcal{H})$ is any (m, J)-representation of C, then either $\mu = \tau$ or $\mu = \tau\{p\}$.*

To see this, note that by Lemma 6.12, either $\mathcal{H}^+(q) = J$ or $\mathcal{H}^+(q) = I$. Also $\sigma(q) = \tau J[q]$ is the compression of $\mu\mathcal{H}^+(q)[q]$. Thus if $\mathcal{H}^+(q) = J$, then $\mu = \tau$, and if $\mathcal{H}^+(q) = I$, then μ must be $\tau\{p\}$.

Now we proceed with constructing an alternative representation of C. Let $A = \mathcal{F}^-(p) \cup \{p\}$. Note that $A \notin \mathcal{F}$ since in the canonical representation, every set in $\mathcal{F}$ is of the form $\mathcal{F}^+(r)$ for some $r \neq p$. Let $\mathcal{G} = \mathcal{F} \cup \{A\}$; then $\mathcal{G}^+(r) = \mathcal{F}^+(r)$ for each $r \neq p$. Thus $(\tau, \mathcal{G})$ would seem to be another (m, J)-representation of C. This is indeed true, except in one case. The problem is that the definition of (m, J) representation requires that $(\tau, \mathcal{G}^+(s))$ be m-admissible for all $s \in J$, which means that $w(\tau) \leq m < w(\tau\mathcal{G}^+(s))$. Now, this is true for all $s \neq p$, since it was true for $(\tau, \mathcal{F})$. However, it is possible that $w(\tau\mathcal{G}^+(p)) \leq m$. This happens if and only if $w(\tau) < m$ and $\mathcal{F}^-(p) = \emptyset$ so that $A = \{p\}$. So we consider two subcases, depending on whether this happens.

Subcase IIa: *$(\tau, \mathcal{G})$ is (m, J)-admissible.* As we have just discussed, this means that either $\mathcal{F}^-(p) \neq \emptyset$ or $\mathcal{F}^-(p) = \emptyset$ and $w(\tau) = m$. Then $(\tau, \mathcal{G})$ is a second (m, J)-representation of $(\tau, \mathcal{F})$; we need to prove that there are no others.

If $(\mu, \mathcal{H})$ is an (m, J)-flag representation of C, then let ν, λ, B be as in Lemma 6.12. Then $B = \{p\}$ or $B = \emptyset$. We claim $B = \emptyset$. If $B = \{p\}$, then $w(\mu) = w(\tau) + 1$ and so the (m, J)-admissibility of $(\mu, \mathcal{H})$ requires $w(\tau) < m$ and so $\mathcal{F}^-(p) \neq \emptyset$. Let $s \in \mathcal{F}^-(p)$; then $\mathcal{F}^+(s) \subseteq \mathcal{F}^-(p)$. But $\tau\{p\}\mathcal{H}^+(s)$ must compress to $\tau\mathcal{F}^+(s)$, which is impossible since $p \notin \mathcal{F}^+(s)$.

Thus $B = \emptyset$, and so $\mathcal{H}^+(q) = \mathcal{F}^+(q) = J$, which means by Proposition 6.14 that $\mu = \tau$. We now need to show that $\mathcal{H} = \mathcal{F}$ or $\mathcal{H} = \mathcal{G}$, i.e., $\mathcal{F} \subseteq \mathcal{H} \subseteq \mathcal{G}$. $\mathcal{H}$ must contain $\mathcal{F}$, since $\mathcal{H}^+(r) = \mathcal{F}^+(r)$ for all $r \in I$ and $\mathcal{F}$ was defined only to contain $\emptyset$, J, and the sets $\mathcal{F}^+(r)$ for $r \in I$. If $\mathcal{H}$ is not a subset of $\mathcal{G}$ let D be the minimal member of $\mathcal{H} - \mathcal{G}$ and let D_0 be the largest subset of D in $\mathcal{G}$. Choose $x \in D - D_0$ and note that $x \neq p$, since $\mathcal{F}^-(p)$ and A are both in $\mathcal{G}$. Then $\mathcal{H}^+(x) = D \neq \mathcal{G}^+(x)$, which contradicts that $(\tau, \mathcal{H})$ and $(\tau, \mathcal{G})$ both are (m, J) representations of C.

Subcase IIb: *$(\tau, \mathcal{G})$ is not (m, J)-admissible.* This means that $\mathcal{F}^-(p) = \emptyset$ and $w(\tau) < m$. Thus $(\tau\{p\}, \mathcal{E})$ is another (m, J)-flag representation, where $\mathcal{E}$ is obtained from $\mathcal{F}$ by deleting p from each set in $\mathcal{F}$ and adding the set J.

Suppose that $(\mu, \mathcal{H})$ is any (m, J)-flag representation of C; we want to show that $(\mu, \mathcal{H}) = (\tau, \mathcal{F})$ or $(\mu, \mathcal{H}) = (\tau\{p\}, \mathcal{E})$. Let ν, λ, B be as in Lemma 6.12. Once again we have either $\mu = \tau$ and $B = \emptyset$ or $\mu = \tau\{p\}$ and $B = \{p\}$.

In the case $\mu = \tau$, we also have that $\mathcal{F}^+(r) = \mathcal{H}^+(r)$ for all $r \neq p$. We claim that $\mathcal{F}^+(p) = \mathcal{H}^+(p)$, which would imply that $\mathcal{F} = \mathcal{H}$. To see the claim, observe that $\mathcal{F}^+(p)$ is the smallest nonempty set of $\mathcal{F}$, and it belongs to $\mathcal{H}$ since $\mathcal{F} \subseteq \mathcal{H}$. Thus it suffices to show that $\mathcal{H}$ contains no smaller set. $\mathcal{H}$ cannot contain $\{p\}$ since $w(\tau p) \leq m$ would violate m-admissibility of $(\tau, \mathcal{H})$. $\mathcal{H}$ cannot contain any other subset of $\mathcal{H}$ because $\mathcal{H}^+(r) = \mathcal{F}^+(r)$ for all $r \neq p$. Thus $\mathcal{F} = \mathcal{H}$, and $(\mu, \mathcal{H}) = (\tau, \mathcal{F})$.

In the case that $\mu = \tau\{p\}$, for any $r \neq q$ we must have that $\tau\mathcal{F}^+(r)$ and $\tau\{p\}\mathcal{H}^+(r)$ compress to the same vertex of C. Then for every $r \neq p$, $\mathcal{H}^+(r) = \mathcal{F}^+(r) - \{p\} = \mathcal{E}^+(r)$. We also have $\mathcal{H}^+(p) = \mathcal{E}^+(p) = J$ since $J - \{p\}$ is a member of both of them. From Proposition 6.11, $\mathcal{H} = \mathcal{E}$, and thus $(\mu, \mathcal{H}) = (\tau\{p\}, \mathcal{E})$. □

This completes the proof of Lemma 6.10 which, as explained, completes the proof that G_m is a triangulation graph and thus completes the proof of the main theorem.

Appendix. The topology of knowable sets. In this appendix, we look more closely at the structure of the collection of knowable sets, $\mathcal{K} = \{K | K \subseteq \hat{\Sigma} \text{ is a knowable set}\}$. In particular, we prove that $\mathcal{K}$ defines a compact Hausdorff topological space on $\hat{\Sigma}(P)$. Along the way we give two characterizations of this space: (i) we give a nice basis for $\mathcal{K}$, and (ii) we show that $\mathcal{K}$ is the quotient of the Cantor topology on $\Sigma(P)$ with respect to the compression map.

Notions from point set topology are briefly reviewed as needed. For more details, see [24].

A.1. $\mathcal{K}$ is a Hausdorff topology. Recall that formally, a topological space on a set X is a collection $\mathcal{U}$ of subsets of X that includes $\emptyset$ and X and is closed under arbitrary union and finite intersection. The members of $\mathcal{U}$ are the *open sets* of the topology, and complements of members of $\mathcal{U}$ are the *closed sets* of the topology.

THEOREM A.1. *$\mathcal{K}$ is a topology on the set $\hat{\Sigma}$.*

Proof. We observed in section 2.8 that $\emptyset$ and $\hat{\Sigma}$ are both in $\mathcal{K}$. We need to show that the union of an arbitrary collection of knowable sets is knowable and the intersection of a finite collection of knowable sets is knowable.

Let $\{K^i\}_{i\in\Lambda}$ be an arbitrary collection of knowable sets. We will show that $K^\cup = \cup_{i\in\Lambda} K^i$ is a knowable set. Let (Π^i, d^i) be an acceptor for K_i and V^i be the set of values that can be written to the registers by this protocol. We exhibit an acceptor $(\Pi^\cup, d)$ for $K^\cup$. Informally the protocol simulates all the protocols in the above collection in parallel and a processor writes the accept value d when it sees that at least one of accept values d^i has been written.

More formally protocol $\Pi^\cup$ is defined as follows. The set of processor states S consists of the (possibly infinite) product set $\prod_{i\in\Lambda}(S^i)$ together with a special state $s^\star$. Thus a state value s is either $s^\star$ or a tuple $(s^i | i \in \Lambda)$, where $s^i \in S^i$. (A state value of $s^\star$ will mean that a processor is ready to write the accept value d.) The initial state e_p is the tuple $(e_p^i | i \in \Lambda)$. The set of write values V is the product set $\prod_{i\in\Lambda} V^i$ together with the accept value d. If no processor has ever written d, then the contents of shared memory $\vec{l}$ can be viewed as a tuple $(\vec{l}^{\,i} : i \in \Lambda)$, where $\vec{l}^{\,i}$ corresponds to the run of Π^i. The write map w is defined as $w(s) = d$ if $s = s^\star$ and otherwise $w(s)$ is the tuple $(w^i(s^i) : i \in \Lambda)$. The state update map u is defined as $u(s, \vec{l}) = s^\star$ if d appears in $\vec{l}$ or there is at least one $i \in \Lambda$ such that d^i appears in $\vec{l}^{\,i}$; otherwise $u(s, \vec{l}) = (u^i(s^i, \vec{l}^{\,i}) : i \in \Lambda)$. It is easy to see that the accept value d is written by $\Pi^\cup$ on schedule σ if and only if there is an $i \in \Lambda$ such that d^i is written by Π^i on σ. Thus the set $K^\cup$ is knowable.

Next we want to show that the intersection of a finite collection of knowable sets is knowable. As above, let $\{K^i\}_{i\in\Lambda}$ be a collection of knowable sets and (Π, d^i) be acceptors. We define a protocol $\Pi^\cap$ by a minor modification of $\Pi^\cup$. The only difference is in the state update map u. The condition for a processor to enter state $s^\star$ is either that some processor has written d or *for every* $i \in \Lambda$, d_i appears in $\vec{l}^{\,i}$.

It is easy to see that if the accept value d is written by $\Pi^\cap$ on schedule σ, then for all $i \in \Lambda$, d^i is written by Π^i on σ and thus $K(\Pi^\cap, d)$ is a subset of $K^\cap$. The reverse containment holds if Λ is finite (although it need not hold if Λ is infinite; see below). For $\sigma \in K^\cap$ let j^i be the index of the block of σ in which d^i is first written and let j be the maximum of the j^i. Then any processor taking a step subsequent to block j will see all of the decision values d^i and thus move to state $s^\star$. At least one such

processor will take another step and will thus write d. ∎

Observe that this final argument fails when the collection $\{K^i\}$ is infinite because the index j may not be defined. As an example, let K^i be the set of compressed schedules where processor p takes at least i steps, and let i range over the positive integers.

Next, recall that a topological space $(X,\mathcal{U})$ is a *Hausdorff space* if for any two distinct points $x, y \in X$ there are disjoint open sets U_x and U_y with $x \in U_x$ and $y \in U_y$.

LEMMA A.2. $(\Sigma,\mathcal{K})$ *satisfies the Hausdorff condition.*

Proof. In this case the Hausdorff condition means that if σ and ϕ are distinct compressed schedules, then there is a pair of acceptors (Π, d) and (Φ, d') such that Π accepts σ and Φ accepts ϕ and no schedule is accepted by both. We will take Π and Φ to be a minor modification of the counting protocol: when processor p writes for the ith time, it writes (T,p), where T is its current tally vector (instead of just T).

Now we need to choose the accepting values for Φ and Π, which will be of the form (T,p). Since σ and ϕ are distinct compressed schedules, Theorem 2.12 implies that their tally records must be different. Apply Lemma 2.5. Under the first conclusion of this lemma, there is a processor p and a positive integer i such that p takes at least i steps in both schedules and tally vectors $Count_{p,i}(\phi)$ and $Count_{p,i}(\phi)$ are different. Thus take $d = (Count_{p,i}(\sigma), p)$ and $d' = (Count_{p,i}(\phi), p)$. Note that for any schedule ρ, at most one of d and d' can appear in its public tally since both can appear only in the list of processor p, and that list can contain only one vector that has an $i-1$ in position p, while both of these vectors have an $i-1$ in that position.

Under the second conclusion of Lemma 2.5 there is a pair of crossing vectors v and w such that v appears in the public tally corresponding to σ and w appears in the public tally corresponding to ϕ. Let q be a processor that writes v during σ and r be a processor that writes w during ϕ. Let $d = (v,q)$ and $d' = (w,r)$; the fact that v and w are crossing ensures that no schedule can be accepted by both protocols. ∎

A.2. A basis for $\mathcal{K}$. A basis for a topology $(X,\mathcal{U})$ is a collection $\mathcal{B}$ of open sets with the property that every open set is a union of members of $\mathcal{B}$. Equivalently, $\mathcal{B}$ is a basis if for any point x and open set U containing x, there is a $B \in \mathcal{B}$ such that $x \in B \subseteq U$.

Recall that for a fragment τ, the set Q_τ is the set of schedules that are quasi extensions of τ, and $\hat{Q}_\tau = Q_\tau \cap \hat{\Sigma}(P)$. Then Example 2.21 and Proposition 2.25 imply the following.

THEOREM A.3. *The set $\{\hat{Q}_\tau : \tau$ a fragment$\}$ is a basis for the knowable set topology.*

A.3. Representing $\mathcal{K}$ as a quotient topology. Let $(X,\mathcal{U})$ be a topological space and $f : X \longrightarrow Y$ be any surjective map. It is easily checked that the collection $\mathcal{U}/f = \{W \subseteq Y : f^{-1}(W) \in \mathcal{U}\}$ defines a topology on Y, called the *quotient* of $(X,\mathcal{U})$ by f. Here we represent the knowable set topology on $\hat{\Sigma}(P)$ as a quotient of a simple topology on $\Sigma(P)$.

For a fragment τ, let B_τ be the set of all schedules that have τ as a prefix. Let $(\Sigma(P),\mathcal{S})$ be the topology whose open sets are unions of sets B_τ. (This is called the *Cantor topology* on $\Sigma(P)$.)

Consider the map $f : \Sigma \longrightarrow \hat{\Sigma}$, where $f(\sigma) = \hat{\sigma}$. In this case, the quotient topology $\mathcal{R} = \mathcal{S}/f$ is defined on $\hat{\Sigma}$ and is given by $\mathcal{R} = \{U \subset \hat{\Sigma} | f^{-1}(U) \in \mathcal{S}\}$.

THEOREM A.4. $\mathcal{K}$ *and* $\mathcal{R}$ *define the same topology on* $\hat{\Sigma}$.

Proof. (1) $\mathcal{K} \subseteq \mathcal{R}$. Since $\{\hat{Q}_\tau | \tau \in \Phi(P)\}$ is a basis for $\mathcal{K}$, it suffices to show that for each fragment τ, $\hat{Q}_\tau$ is in $\mathcal{R}$, i.e., that $f^{-1}(\hat{Q}_\tau)$ is open in the Cantor topology. For this, it suffices to show that if $\sigma \in f^{-1}(\hat{Q}_\tau)$, then there is a fragment μ so that $\sigma \in B_\mu \subseteq f^{-1}(\hat{Q}_\tau)$. Since σ is in $f^{-1}(\hat{Q}_\tau)$ it is a quasi extension of τ, which by Theorem 2.16 means that its public tally is an extension of the public tally of τ. Since τ is finite, there is a prefix μ of σ such that the public tally of μ extends the public tally of τ. This implies that any schedule in B_μ is a quasi extension of τ, i.e., $B_\mu \subseteq f^{-1}(\hat{Q}_\tau)$.

(2) $\mathcal{R} \subseteq \mathcal{K}$. Let $S \subset \hat{\Sigma}$ be a set such that $f^{-1}(S)$ is an open set in the Cantor topology. We will prove that S is a knowable set. Let $\sigma \in S$ be arbitrary. It suffices to show that there is a knowable set K containing σ such that $K \subset S$.

Let $f^{-1}(\sigma) = \{\sigma^1, \ldots, \sigma^k\}$, which is a finite set by Proposition 2.10. For each $\sigma^i \in f^{-1}(\sigma)$ let B_{ρ_i} be a basis set in the Cantor topology such that $\sigma^i \in B_{\rho_i} \subset f^{-1}(S)$.

Choose a prefix ρ of σ that has greater weight (total number of steps) than each of the ρ_i and also contains all of the steps of the faulty processors of σ. We write $\sigma = \rho\chi$, where Active(χ) is equal to N, the set of nonfaulty processors of σ. Hence, ρN is compressed since $\rho\chi$ is. By Lemma 2.15, $\sigma \in \hat{Q}_{\rho N}$. We claim that $\hat{Q}_{\rho N} \subseteq S$, which will complete the proof.

Let γ be an arbitrary schedule in $\hat{Q}_{\rho N}$; we show that $\gamma \in S$. For this it suffices to find a j such that $\gamma \in B_{\rho_j}$, i.e., ρ_j is a prefix of γ. By Theorem 2.16 there is a prefix τ of γ, a schedule ϕ, and a subset U of Active(ϕ) such that $\gamma = \tau\phi$ and $\widehat{\tau U} = \rho N$. Since $\widehat{\tau U} = \rho N$ we must have $U \subseteq N$ and $\tau = \beta\zeta$, where ζ consists of some sequence of disjoint blocks whose union is $U - N$ and the last block of β has nonempty intersection with N. Thus $\widehat{\beta N} = \widehat{\tau U} = \rho N$.

We now claim that $\widehat{\beta\chi} = \sigma$. Now, since σ is compressed, so is χ. It is easy to see that a block is hidden in $\beta\chi$ if and only if it is hidden inside β within the fragment βN, and since the compression of βN is ρN, we have $\widehat{\beta\chi} = \rho\chi = \sigma$. Hence $\beta\chi \in f^{-1}(\sigma)$, and $\beta\chi \in B_{\rho_j}$ for some j. Since β and ρ have the same weight, which is at least the weight of ρ_j, we have that ρ_j is a prefix of β. Therefore $\gamma = \beta\zeta\phi \in B_{\rho_j}$. ∎

A topological space $(X, \mathcal{U})$ is *compact* if for any collection of open sets whose union is X there is a finite subcollection whose union is X. We remark that since the Cantor topology is known to be compact, and a quotient of a compact topology is compact, we have the following.

COROLLARY A.5. *The topological space* $(\hat{\Sigma}, \mathcal{K})$ *is compact.*

Acknowledgments. We would like to express our appreciation to Soma Chaudhuri for introducing us to this problem and for several enlightening discussions. We thank Shlomo Moran for helpful comments on an earlier draft. We thank two anonymous referees for numerous helpful suggestions. Thanks to Shiyu Zhou and Sri Divakaran for help in making several of the figures.

REFERENCES

[1] P. S. ALEKSANDROV, *Combinatorial Topology*, Graylock Press, Rochester, NY, 1956.

[2] Y. AFEK, H. ATTIYA, D. DOLEV, E. GAFNI, M. MERRITT, AND N. SHAVIT, *Atomic snapshots*, J. ACM, 40 (1993), pp. 873–890. A preliminary version appeared as *Atomic snapshots of shared memory* in Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, 1990, pp. 1–13.

[3] J. ANDERSON, *Composite registers*, Distrib. Comput., 6 (1993), pp. 141–154. A preliminary version appeared in Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, 1990, pp. 15–29.

[4] J. Aspnes and M. Herlihy, *Wait-free data structures in the asynchronous PRAM model*, in Proceedings of the Second Annual Symposium on Parallel Algorithms and Architectures, Crete, Greece, 1990, pp. 340–349.

[5] H. Attiya, N. Lynch, and N. Shavit, *Are wait-free algorithms fast?*, J. ACM, 41 (1994), pp. 725–763. A preliminary version appeared in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 55–64.

[6] O. Biran, S. Moran, and S. Zaks, *A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor*, J. Algorithms, 11 (1990), pp. 420–440. A preliminary version appeared in Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, 1988, pp. 263–275.

[7] E. Borowsky and E. Gafni, *Generalized FLP impossibility result for t-resilient asynchronous computations*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 91–100.

[8] E. Borowsky and E. Gafni, *Immediate atomic snapshots and fast renaming*, in Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing, Ithaca, NY, 1993, pp. 41–51.

[9] H. Brit and S. Moran, *Wait-freedom vs. bounded wait-freedom in public data structures*, J. UCS, 2 (1996), pp. 2–19.

[10] S. Chaudhuri, *More choices allow more faults: Set consensus problems in totally asynchronous systems,* Inform. and Comput., 105 (1993), pp. 132–158. A preliminary version appeared as *Agreement is harder than consensus: Set consensus problems in totally asynchronous systems* in Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, 1990, pp. 311–324.

[11] S. Chaudhuri, M. Herlihy, N. Lynch, and M. Tuttle, *A tight lower bound for k-set agreement*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1994, pp. 206–215.

[12] R. Cole and O. Zajicek, *The expected advantage of asynchrony*, J. Comput. System Sci., 51 (1995), pp. 286–300. A preliminary version appeared in Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, 1990, pp. 85–94.

[13] D. Dolev and N. Shavit, *Bounded concurrent time-stamps*, SIAM J. Comput., 6 (1997), pp. 418–455. A preliminary version appeared as *Bounded concurrent time-stamp systems are constructible!* in Proceedings of the 21th Annual ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 454–465.

[14] R. Fagin, Y. J. Halpern, Y. Moses, and M. Vardi, *Reasoning about Knowledge*, MIT Press, Cambridge, UK, 1995.

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

[16] M. Herlihy, *Wait-free synchronization*, ACM Trans. Programming Languages Systems, 13 (1991), pp. 124–149.

[17] M. Herlihy and N. Shavit, *The topological structure of asynchronous computability*, J. ACM, to appear. A preliminary version appeared as *The asynchronous computability theorem for t-resilient tasks* in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 111–120.

[18] M. Herlihy and N. Shavit, *A simple constructive computability theorem for wait-free computation*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, Montreal, Quebec, Canada, 1994, pp. 243–252.

[19] A. Israeli and Ming Li, *Bounded time-stamps*, Distrib. Comput., 6 (1993), pp. 205–209. A preliminary version appeared in Proceedings of the 28th Annual Symposium on Foundations of Computer Science, Los Angeles, CA, 1987, pp. 371–382.

[20] L. Lamport, *On Interprocess Communication. Part* I: *Basic Formalism, Part* II: *Algorithms*, Distrib. Comput., 1 (1986), pp. 77–101.

[21] M. C. Loui and H. H. Abu-Amara, *Memory requirements for agreement among unreliable asynchronous processes*, in Adv. Comput. Res. 4, JAI Press, Greenwich, CT, 1987, pp. 163–183.

[22] C. Martel, A. Park, and R. Subramonian, *Work-optimal asynchronous algorithms for shared memory parallel computers*, SIAM J. Comput., 21 (1992), pp. 1070–1099. A preliminary version appeared as *Asynchronous PRAMs are (almost) as good as synchronous PRAMs* in Proceeding of the 31st Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 590–599.

[23] S. Moran and Y. Wolfstahl, *Extended impossibility results for asynchronous complete networks*, Inform. Process. Lett., 26 (1987), pp. 145–151.

[24] J. R. Munkres, *Topology: A First Course*, Prentice–Hall, Englewood Cliffs, NJ, 1975.
[25] G. L. Peterson and J. E. Burns, *Concurrent reading while writing* II: *The multiwriter case*, in Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, Los Angeles, CA, 1987.
[26] G. M. Reed, A. W. Roscoe, and R. F. Wachter, eds., *Topology and Category Theory in Computer Science*, Clarendon Press, Oxford, UK, 1991.
[27] S. Vickers, *Topology Via Logic*, Cambridge Tracts in Theoret. Comput. Sci. 6, Cambridge University Press, Cambridge, UK, 1997.