List of important publications in concurrent, parallel, and distributed computing

These papers provide a breadth of information about concurrent, parallel, and distributed computing that is generally useful and interesting from a computer science perspective.

Contents

The Topological Structure of Asynchronous Computability

MAURICE HERLIHY

Brown University, Providence, Rhode Island

AND

NIR SHAVIT

Tel-Aviv University, Tel-Aviv Israel

Abstract. We give necessary and sufficient combinatorial conditions characterizing the class of decision tasks that can be solved in a wait-free manner by asynchronous processes that communicate by reading and writing a shared memory.

We introduce a new formalism for tasks, based on notions from classical algebraic and combinatorial topology, in which a task's possible input and output values are each associated with high-dimensional geometric structures called simplicial complexes. We characterize computability in terms of the topological properties of these complexes. This characterization has a surprising geometric interpretation: a task is solvable if and only if the complex representing the task's allowable inputs can be mapped to the complex representing the task's allowable outputs by a function satisfying certain simple regularity properties.

Our formalism thus replaces the "operational" notion of a wait-free decision task, expressed in terms of interleaved computations unfolding in time, by a static "combinatorial" description expressed in terms of relations among topological spaces. This allows us to exploit powerful theorems from the classic literature on algebraic and combinatorial topology. The approach yields the first impossibility results for several long-standing open problems in distributed computing, such as the "renaming" problem of Attiya et al., and the "k-set agreement" problem of Chaudhuri.

Preliminary versions of these results appeared as HERLIHY, M. P., AND SHAVIT, N. 1993. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (San Diego, Calif., May 16–18). ACM, New York, pp. 111–120, and HERLIHY, M. P., AND SHAVIT, N. 1994. A simple constructive computability theorem for wait-free computation. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing* (Montreal, Que., Canada, May 23–25). ACM, New York, pp. 243–252.

The work of M. Herlihy was supported by NSF grant DMS 95-05949.

The work of N. Shavit was supported by NSF grant CCR 95-20298 and Israeli Academy of Science grant Number 0361-88.

Authors' addresses: M. Herlihy, Computer Science Department, Brown University, Box 1910, Providence, R. I. 02912; N. Shavit, Computer Science Department, Tel-Aviv University, Tel-Aviv 69978, Israel.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*computability theory*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*parallelism and concurrency*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Algebraic topology, asynchronous distributed computation, decision tasks, distributed computing, homology, simplicial complex, wait-free algorithms

1. *Introduction*

Modern multiprocessors, whether they communicate by message-passing or through shared memory, are inherently *asynchronous*: processes can be halted or delayed without warning by cache misses, interrupts, or scheduler pre-emption. A *task* in an *asynchronous system* is a problem where each process starts with a private input value, communicates with the others, and halts with a private output value. A *protocol* is a program that solves the task. In asynchronous systems, it is desirable to design protocols that are *wait-free*: any process that continues to run will halt with an output value in a fixed number of steps, regardless of delays or failures by other processes.

Under what circumstances does a task have a wait-free protocol? In this paper, we give the first completely combinatorial characterization of the circumstances under which tasks have wait-free protocols in shared read/write memory. We show that any task and any protocol can be associated with a pair of high-dimensional geometric structures called *simplicial complexes*. A protocol solves a task if and only if their simplicial complexes can be mapped to one another by a function satisfying certain simple regularity properties. Our main theorem gives necessary and sufficient conditions for such a map to exist.

Although our characterization is quite general, it has concrete applications. In particular, it yields the first impossibility results for several long-standing open problems in distributed computing, including the *renaming* problem of Attiya et al. [1990] with a small number of names, and the *set agreement* problem of Chaudhuri [1990]. It is also a building block in other characterizations such as Afek and Stupp's [1993] characterization of the effect of register size on the power of multiprocessor synchronization operations.

Informally speaking, impossibility is demonstrated as follows: Our theorem implies that there exists a map from the protocol complex to the task complex that preserves certain topological properties. We can establish that no map exists by showing that the complexes are topologically "incompatible," in much the same way that classical algebraic topology uses topological invariants to prove that two spaces cannot be homeomorphic, that is, cannot be continuously mapped each to the other. In particular, we show that the simplicial complex associated with any wait-free protocol using read/write memory has a remarkable topological property: it has no "holes" in any dimension. We exploit this simple property to derive our impossibility results.

In a fundamental paper, Fischer et al. [1985] showed that there exists a simple task that cannot be solved in a message-passing system if even one process may fail by halting (or may be infinitely slow). This result showed that the notion of "asynchronous computability" differs in important ways from conventional notions of computability (such as sequential "Turing" computability). It led to the

creation of a highly active research area, the full scope of which is surveyed in recent book by Lynch [1996].

A first step toward a systematic characterization of asynchronous computability was taken in 1988 by Biran et al. [1988] who gave a graph-theoretic characterization of the tasks that could be solved by a message-passing system in the presence of a single failure. Although the problem subsequently received considerable attention, it proved difficult to extend such graph-theoretic approaches to encompass more than a single failure. Even the problem of fully characterizing specific tasks like *renaming* [Attiya et al. 1990] and *set agreement* [Chadhuri 1990] remained unresolved. Chor and Moscovici [1980] later provided a graph-theoretic characterization of tasks solvable in a system where the $n + 1$ processes can solve $(n + 1)$-process consensus (either deterministically or randomized).

In 1993, three research teams—Borowsky and Gafni [1993], Saks and Zaharoglou [1993], and the current authors [1993] independently derived lower bounds for the *k-set agreement* problem of Chaudhuri. The proof of Borowsky and Gafni [1993] is based on a powerful simulation method that allows N-process protocols to be executed by fewer processes in a resilient way. Both Saks and Zaharoglou [1993] and the current authors [1993] apply notions and techniques from mainstream combinatorial topology. Saks and Zaharoglou construct an elegant model that casts processors' collective knowledge of the unfolding computation as a topological space, and apply a variant of the Brouwer fixed point theorem [Munkres 1984] to derive impossibility of wait-free set agreement. This technique appears to be specific to set agreement.

In contrast, our work [Herlihy and Shavit 1993; 1994] focused on general properties of the model of computation rather than on properties of specific tasks. We introduced a new formalism based on simplicial complexes and homology, notions taken from undergraduate-level algebraic topology. The new formalism replaces the popular "operational" notion of a wait-free decision task, expressed in terms of interleaved computations unfolding in time, by a static "combinatorial" description expressed in terms of relations among simplicial complexes. Simplicial complexes are a natural generalization of graphs. They provide a notion of dimensionality, absent in earlier graph-theoretic models, that captures in a natural way the effects of multiple failures. A further advantage of this model is that it becomes possible to apply standard results from mainstream mathematics to distributed computation.

The paper is organized as follows: Section 2 provides the details of our new topological framework for asynchronous computation. Section 3 presents the statement of our main theorem and a collection of example results derived from it, including the impossibility of wait-free *set agreement*. Sections 5 and 4 respectively provide the proofs of the "if" and "only if" parts of our theorem. We conclude the paper with a proof of the impossibility of solving the *renaming* problem with a small number of names.

2. *Model*

We begin with an informal synopsis of our model, in which $N = n + 1$ sequential threads of control, called processes, cooperate to solve a decision task. In a decision task, each process starts with a private input value, and halts with a

private output value. For example, in the well-known *binary consensus* task, the processes have binary inputs, and must agree on some process's input as their common output [Fischer et al. 1985]. A protocol is a program that solves a decision task. A protocol is wait-free if it guarantees that every nonfaulty process will finish in a bounded number of steps, no matter how many processes fail.

This paper considers protocols in which processes communicate by reading and writing variables in shared memory. The literature encompasses a variety of shared-memory models; fortunately they are all equivalent in the sense that any one can be implemented in a wait-free manner from any other. From the simplest single-bit, single-reader, and single-writer variables, one can construct multi-bit, multi-reader variables (see Lynch's survey [Lynch 1996]). From these variables, in turn, one can implement an *atomic snapshot memory*: an array where each P_i updates (writes) array element i, and any process can instantaneously scan (atomically read) the entire array [Afek et al. 1990; Anderson 1990]. It is thus convenient to assume that processes communicate via atomic snapshot memory.

In the remainder of this section, we restate the model in more formal terms, and introduce the mathematical concepts underlying our main theorem and its proof.

2.1. Decision Tasks. We now define decision tasks more precisely. We start with the notion of a vector, which describes the input/output behavior of finite executions. Let D_I and D_O be data types, possibly the same, called the *input* and *output* data types.

Definition 2.1. An $(n + 1)$-process *input vector* $\vec{I}$ (respectively, *output vector* $\vec{O}$) is an $(n + 1)$-component vector, each component of which is either a value of type D_I (respectively, D_O), or the distinguished value $\perp$. At least one component of $\vec{I}$ (respectively, $\vec{O}$) must be different from $\perp$.

We denote the ith component of input vector $\vec{I}$ by $\vec{I}[i]$, and similarly for output vectors. An input vector $\vec{I}$ represents a possible assignment of input values to processes: if $\vec{I}[i]$ is an input value v, then v is P_i's input at the start of the execution, while if $\vec{I}[i]$ is $\perp$, then P_i does not participate in that execution: it has no input and takes no steps. Similarly, an output vector $\vec{O}$ represents a possible choice of output values by processes: if $\vec{O}[i]$ is an output value v, then v is P_i's output chosen during that execution, while if $\vec{I}[i]$ is $\perp$, then P_i does not choose an output in that execution. Vectors thus describe the input/output behavior of finite executions in which some subset of processes participate.

Definition 2.2. A *participating index* (*process*) in an input vector is one whose value is distinct from $\perp$.

Definition 2.3. Vector $\vec{U}$ *matches* $\vec{V}$ if, for $0 \leq i \leq n$, $\vec{U}[i] = \perp$ if and only if $\vec{V}[i] = \perp$.

Matching vectors have the same set of participating indexes. We are often concerned with executions that are prefixes of a given execution.

Definition 2.4. Vector $\vec{U}$ is a *prefix* of $\vec{V}$ if, for $0 \leq i \leq n$, either $\vec{U}[i] = \vec{V}[i]$ or $\vec{U}[i] = \perp$.

$\vec{I}$	$\Delta(\vec{I})$
$(0,\perp,\perp)$	$(0,\perp,\perp),(1,\perp,\perp),(2,\perp,\perp)$
$(\perp,0,\perp)$	$(\perp,0,\perp),(\perp,1,\perp),(\perp,2,\perp)$
$(\perp,\perp,0)$	$(\perp,\perp,0),(\perp,\perp,1),(\perp,\perp,2)$
$(0,0,\perp)$	$(0,1,\perp),(1,0,\perp),(0,2,\perp),(2,0,\perp),(2,1,\perp),(1,2,\perp)$
$(0,\perp,0)$	$(0,\perp,1),(1,\perp,0),(0,\perp,2),(2,\perp,0),(2,\perp,1),(1,\perp,2)$
$(\perp,0,0)$	$(\perp,0,1),(\perp,1,0),(\perp,0,2),(\perp,2,0),(\perp,2,1),(\perp,1,2)$
$(0,0,0)$	$(0,1,2),(0,2,1),(1,0,2),(1,2,0),(2,0,1),(2,1,0)$

FIG. 1. The Unique-Id task.

If a prefix has an entry distinct from $\perp$, then it agrees with the corresponding entry in the original.

Definition 2.5. A set V of vectors is *prefix-closed* if for all $\vec{V} \in V$, every prefix $\vec{U}$ of $\vec{V}$ is in V.

We use prefix-closed sets of vectors to characterize the legitimate sets of input and output value assignments. If the set of input vectors is prefix-closed, then any legitimate assignment of input values remains legitimate if fewer processes participate. If the set of output vectors is prefix-closed, then any legitimate choice of output values remains legitimate if fewer processes decide.

Definition 2.6. Let I and O be prefix-closed sets of input and output vectors. A *task specification* is a relation $\Delta \subseteq I \times O$, carrying each input vector to a non-empty subset of *matching* output vectors.

Let $\Delta(\vec{I})$ denote the set of vectors $\vec{O}$ in O such that $(\vec{I}, \vec{O}) \in \Delta$. For a given input vector $\vec{I}$, the set $\Delta(\vec{I})$ is just the set of legitimate output vectors corresponding to the inputs $\vec{I}$. Because task specifications are typically nondeterministic, $\Delta(\vec{I})$ typically contains multiple vectors.

We are now ready to give a precise definition of decision tasks.

Definition 2.7. A *decision task* $\langle I, O, \Delta \rangle$ is a tuple consisting of a set I of input vectors, a set O of output vectors, and a task specification Δ relating these two sets.

This class of decision tasks includes all linearizable *one-time objects*, that is linearizable objects [Herlihy and Wing 1990] that permit at most one operation per process. The model captures the intuitive notion of "order of events in time" through the use of participating processes. For example, although the following tasks have the same sets of input and output vectors, they have a very different structure.

Unique-id. Each participating process $P_i \in \{0, \ldots, n\}$ has an input $x_i = 0$ and chooses an output $y_i \in \{0, \ldots, n\}$ such that for any pair $P_i \neq P_j$, $y_i \neq y_j$.

Fetch-And-Increment. Each participating process $P_i \in \{0, \ldots, n\}$ has an input $x_i = 0$ and chooses a unique output $y_i \in \{0, \ldots, n\}$ such that (1) for some participating index i, $y_i = 0$, and (2) for $1 \leq k \leq n$, if $y_i = k$, then, for some $j \neq i$, $y_j = k - 1$.

The tables in Figure 2 show the task specifications for *Unique-id* and *Fetch-And-Increment*. Notice that *Unique-Id* allows identifiers to be assigned statically,

$\vec{I}$	$\Delta(\vec{I})$
$(0, \perp, \perp)$	$(0, \perp, \perp)$
$(\perp, 0, \perp)$	$(\perp, 0, \perp)$
$(\perp, \perp, 0)$	$(\perp, \perp, 0)$
$(0, 0, \perp)$	$(0, 1, \perp), (1, 0, \perp)$
$(0, \perp, 0)$	$(0, \perp, 1), (1, \perp, 0)$
$(\perp, 0, 0)$	$(\perp, 0, 1), (\perp, 1, 0)$
$(0, 0, 0)$	$(0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1), (2, 1, 0)$

FIG. 2. The Fetch-And-Inc task.

while *Fetch-And-Increment* effectively requires that they be assigned dynamically in increasing order. We will see that the first task has a trivial wait-free solution, while the second has no solution in read/write memory if one or more processes can fail.

2.2. OBJECTS, PROCESSES, AND PROTOCOLS. Formally, we model objects, processes, and protocols using a simplified form of the I/O automaton formalism of Lynch and Tuttle [1988]. An I/O *automaton* is a nondeterministic automaton with a finite or infinite set of *states*, a set of *input events*, a set of *output events*, and a transition relation given by a set of *steps*, each defining a state transition following a given event. An *execution* of an I/O automaton is an alternating sequence of states and enabled events, starting from some initial state. An *execution fragment* is a subsequence of consecutive states and events occurring in an execution. For simplicity, we will use the term, *execution*, to mean either execution or execution fragment, the appropriate term being clear from context. All executions considered in this paper are finite. An automaton *history* is the subsequence of events occurring in an execution. Automata can be composed by identifying input and output events in the natural way (details can be found in Lynch and Tuttle [1988]).

An *object* X is an automaton with input events CALL(P, v, X, T) and output event RETURN(P, v, X, T) where P is a process id, v is a value, and X an object, and T is a type. A *process* P is an automaton with output events CALL(P, v, X, T), and FINISH(P, v) and input events RETURN(P, v, X, T) and START(P, v). A RETURN event *matches* an earlier CALL event in a history if the two events have the same type, name, and process id. An *operation* is a matching pair of CALL and RETURN events.

A *read/write memory* object M is an automaton with input events READ(P, a) and WRITE(P, a, v)[1] and output event RETURN(P, v). The read/write memory model we use in this paper is the standard *atomic snapshot memory*: an array a where each WRITE(P, a, v) is an *update* of array element $a[P]$ to the value v, and each READ(P, a) is an instantaneous *scan* operation returning the contents of the entire array a (see Afek et al. [1990] and Anderson [1990] for details and Lynch [1996] for a survey of why read/write memory models are all equivalent to atomic snapshot memory.)

A history is *sequential* if each call event is immediately followed by a matching response. (A sequential execution permits process steps to be interleaved, but at

[1] Strictly speaking, READ(P, a) is short for CALL(P, READ, M, a), and similarly for WRITE.

the granularity of complete operations.) If we restrict our attention to sequential histories, then the behavior of the atomic snapshot memory is straightforward: any *scan* operation of array a returns for each element $a[P]$ the value of the last preceding *update* by process P, or $\perp$ if no such update existed. Each history H induces a partial "real-time" order $\prec_H$ on its operations: $op_0 \prec_H op_1$ if the output event for op_0 precedes the input event for op_1. Operations unrelated by $\prec_H$ are said to be *concurrent*. If H is sequential, $\prec_H$ is a total order. A concurrent protocol or object is *linearizable* if for every history H, there exists a sequential history G with the same events as H, where $\prec_H \subseteq \prec_G$. Informally, a history is linearizable if it can be mapped to a sequential history by making each operation appear to take effect instantaneously at some point between its call and its response (see Herlihy and Wing [1990] for details). An *atomic snapshot memory* object is one that is linearizable to the sequential object specified above.

A *protocol* $\{P_0, \ldots, P_n; M\}$ is the automaton composed by identifying in the obvious way the events for processes $P_0, \ldots, P_n$ and the memory M. At any point in the execution of a protocol, the state of each process is called its *local state*. The set of local states together with the state of the memory is called the protocol's *global state*. To capture the notion that a process represents a single thread of control, a protocol execution is *well-formed* if every process history (the projection of the history onto the actions of P_i) has a unique START event (generated externally to the protocol), which precedes any CALL or RETURN events, it alternates matching CALL and RETURN events, and has at most one FINISH event. We restrict our attention to well-formed executions.

Definition 2.8. Operations p and q of object X *commute* if, for all sequential histories H and G, $H \cdot p \cdot q \cdot G$ is a history of X if and only if $H \cdot q \cdot p \cdot G$ is a history of X (where "$\cdot$" is the concatenation operator).

In this paper, we use atomic snapshot memory, where *scan* (READ) operations commute with one another, as do memory *update* (WRITE) operations. This property will be shown as fundamental in determining the computational power of read/write memory.

For brevity, we express protocols using pseudo-code, although it is straightforward to translate this notation into automaton definitions.

2.3. SOLVABILITY. We are interested in characterizing when tasks can be solved by processes that are individually equivalent to Turing machines. A process is *active* at a point in an execution if it does not yet have a FINISH event. An active process is *faulty* at that point if it has no output events later in that execution. An execution is *t-faulty* if up to t processes become faulty.

Definition 2.9. A protocol *solves* a decision task in an execution if the following condition holds. Let $\{P_i | i \in U\}$ be the set of processes that have START events, and let $\{u_i | i \in U\}$ be their arguments. Let $\{P_j | j \in V\}$, $V \subseteq U$, be the processes that execute FINISH events, and let $\{v_j | j \in V\}$ be their output values. Let $\vec{I}$ be the input vector with u_i in component i, and $\perp$ elsewhere, and let $\vec{O}$ be the corresponding output vector for the v_j. We require that

(1) no process takes an infinite number of steps without a FINISH event, and
(2) $\vec{O}$ is a prefix of some vector in $\Delta(\vec{I})$.

```
% Code for process i
update(i,a,input_value)      % a[i] := input_value
for round in 1 .. r do
  local_state := scan(a)
  update(i,a,local_state)    % a[i] := local_state
return δ(local_state)
```

FIG. 3. A Wait-Free Protocol in Normal Form.

Informally, the second condition implies that if a protocol solves a task in an execution, the outputs of the nonfaulty processes in any prefix of the execution are consistent with the allowable outputs of the possibly larger set of inputs to the execution as a whole. A protocol for N processes *wait-free solves* a decision task if it solves it in every t-faulty execution where $0 \leq t < N$. We will call such a protocol *wait-free* and henceforth use the term *solves* to mean *wait-free solves*.

It is convenient to assume that any wait-free protocol using read/write memory is expressed in the following *normal form*. The processes share an atomic snapshot memory array a whose $N = n + 1$ elements are all initially $\perp$. Each process has a *local state*, consisting of its input value and the history of values it has so far read from the shared memory. Computation proceeds in a sequence of asynchronous *rounds*, from 0 to some fixed r. In round 0, each process writes its input value to its local variable. In any subsequent round, each P_i executes a sequence of *steps*: (1) it *updates* $a[i]$ to its current local state, and (2) it atomically *scans* the elements of a, appending them to its local state. After r rounds, P_i computes its output value by applying a task-specific *decision map* δ to its final local state. Figure 3 shows a generic protocol in normal form.

Because the set I of input vectors is finite, any kind of wait-free atomic snapshot memory protocol can be expressed in normal form.[2] The memory locations $a[i]$ need only be of bounded size, though for our lower bound proofs we allow them to be unbounded. Without loss of generality, we may assume that each time a process performs an update operation it writes a unique value.

2.4. SIMPLICIAL COMPLEXES. We start with a number of standard technical definitions taken mostly from standard undergraduate textbooks [Munkres 1984; Spanier 1966].

A *vertex* $\vec{v}$ is a point in a high dimensional Euclidean space. A set $\{\vec{v}_0, \ldots, \vec{v}_n\}$ of vertexes is *affinely independent* if and only if the vectors $\vec{v}_1 - \vec{v}_0, \cdots, \vec{v}_n - \vec{v}_0$ are linearly independent.

Definition 2.10. Let $\{\vec{v}_0, \ldots, \vec{v}_n\}$ be an affinely independent set of $n + 1$ vertexes. We define the (geometric) *n-simplex* S spanned by $\vec{v}_0, \ldots, \vec{v}_n$ to be the set of all points x such that $x = \sum_{i=0}^{n} t_i \vec{v}_i$ where $\sum_{i=0}^{n} t_i = 1$ and $t_i \geq 0$ for all i.

For example, a 0-simplex is a vertex, a 1-simplex a line segment, a 2-simplex a solid triangle, and a 3-simplex a solid tetrahedron. For an example of vertexes[3]

[2] If the number of input vectors were unbounded, then the protocol would need an explicit termination test since it could be that there is no bound r on the maximal number of rounds necessary to complete a task.

[3] In the example we name vertexes with names like $P0$ and $Q1$ using a combination of process id (such as P or Q) and a "value" (such as 0 or 1). The reason will become clear in the sequel.

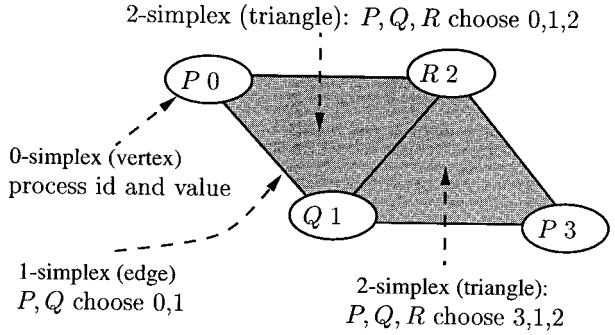

FIG. 4. Vertexes and simplexes.

and simplexes, see Figure 4. For simplicity, we often denote the simplex spanned by a set $\{\vec{v}_0, \ldots, \vec{v}_n\}$ of affinely independent vertexes as $(\vec{v}_0, \ldots, \vec{v}_n)$. The number n is called the *dimension* of the simplex S, and is often denoted by $\dim(S)$. For clarity, we often indicate the dimension of a simplex as a superscript: S^n denotes to the simplex spanned by the vertexes in $\{\vec{v}_0, \ldots, \vec{v}_n\}$.

Any simplex T spanned by a subset of $\{\vec{v}_0, \ldots, \vec{v}_n\}$ is called a *face* of S, denoted $T \subseteq S$. The faces of S different from S itself are called the *proper* faces of S. In Figure 4, the 1-simplex spanned by the vertexes $\{P0, Q1\}$ is a proper face of the 2-simplex $\{P0, Q1, R2\}$. The union of the proper faces of S is called the *boundary* of S, and is denoted $Bd(S)$. The interior of S, denoted $Int(S)$, is defined by the set equation $Int(S) = S - Bd(S)$.

Definition 2.11. Let $S^d = (\vec{s}_0, \ldots, \vec{s}_d)$ be a d-simplex. Define $face_i(S^d)$, the ith face of S^d, to be the $(d - 1)$-simplex $(\vec{s}_0, \ldots, \hat{s}_i, \ldots, \vec{s}_d)$, where the circumflex denotes omission.

As will soon become clear, we will use vertexes to model local process states, and simplexes to model consistent states of multiple processes involved in solving a decision task or in running a protocol in the atomic snapshot model. To model a collection of such states, we need the concept of a geometric simplicial complex, or complex for short.

Definition 2.12. A geometric simplicial complex $\mathcal{K}$ in a Euclidean space is a collection of geometric simplexes such that

—Every face of every simplex of $\mathcal{K}$ is also a simplex of $\mathcal{K}$.

—The intersection of any two simplexes of $\mathcal{K}$ is also a simplex of $\mathcal{K}$.

We consider only finite complexes. The *dimension* of a complex $\mathcal{K}$, denoted $\dim(\mathcal{K})$, is the highest dimension of any of its simplexes, and is sometimes indicated explicitly by a superscript. An n-dimensional complex (or n-complex) is *pure* if every simplex is a face of some n-simplex. Except when noted, all complexes considered in this paper are pure. A simplex S in $\mathcal{K}$ with dimension $\dim(S) = \dim(\mathcal{K})$ is called a *principal* simplex.

Figure 4 shows an example of a pure 2-dimensional complex consisting of the union of the faces of the 2-simplexes $(P0, Q1, R2)$ and $(P3, Q1, R2)$. Both 2-simplexes are principal simplexes in this example.

If $\mathcal{L}$ is a subcollection of simplexes in $\mathcal{K}$ closed under containment and intersection, then $\mathcal{L}$ is a complex in its own right, called a *subcomplex* of $\mathcal{K}$.

The set of simplexes of $\mathcal{K}$ of dimension at most ℓ is a subcomplex of $\mathcal{K}$, called the *ℓ-skeleton* of $\mathcal{K}$, denoted $skel^{\ell}(\mathcal{K})$. The elements of $skel^{0}(\mathcal{K})$ are the *vertexes* of $\mathcal{K}$. For example the 0-skeleton of $\mathcal{K}$ in Figure 4 is just $\{P0, P3, Q1, R2\}$. Similarly, the 1-skeleton of $\mathcal{K}$ is the union of the 0-skeleton and the set of 1-simplexes $\{(P0, Q1), (P0, R2) (R2, Q1) (P3, Q1) (P3, R2)\}$. We now define a way of "adding" simplexes, known as *joining*.

Definition 2.13. Let $S = (\vec{s}_0, \ldots, \vec{s}_p)$ and $T = (\vec{t}_0, \ldots, \vec{t}_q)$ be simplexes whose combined sets of vertexes are affinely independent. Then the *join* of S and T, denoted $S \cdot T$ is the simplex $(\vec{s}_0, \ldots, \vec{s}_p, \vec{t}_0, \ldots, \vec{t}_q)$.

We may extend the notion of joining to complexes as well.

Definition 2.14. If $\mathcal{K}$ and $\mathcal{L}$ are simplicial complexes, not necessarily of the same dimension, then their *join*, denoted $\mathcal{K} \cdot \mathcal{L}$, is the collection of simplexes $\mathcal{K} \cup \mathcal{L} \cup \{S \cdot T | S \in \mathcal{K}, T \in \mathcal{L}\}$.

The join of two complexes $\mathcal{K}$ and $\mathcal{L}$ is a complex in its own right [Munkres 1984]. One useful complex derived using the join operator is the *cone* over $\mathcal{K}$, defined as $\vec{v} \cdot \mathcal{K}$ for some vertex $\vec{v}$ affinely independent of $\mathcal{K}$. Let $|\mathcal{K}|$ be the subset $\bigcup_{S \in \mathcal{K}} S$ of a high-dimensional Euclidean space $\mathbf{R}^{\ell}$, that is, the union of the simplexes of $\mathcal{K}$. Giving each simplex its natural topology as a subspace of $\mathbf{R}^{\ell}$, we topologize $|\mathcal{K}|$ by declaring a subset A of $|\mathcal{K}|$ to be closed if and only if $A \cap S$ is closed for all $S \in \mathcal{K}$. This space is called the *polyhedron* of $\mathcal{K}$. Conversely, $\mathcal{K}$ is called a *triangulation* of $|\mathcal{K}|$. We define the *diameter* of a simplex S to be the maximum Euclidean distance between any pair of points of $|S|$.

Definition 2.15. Two topological subspaces A and B are *homeomorphic* if there exists a one-to-one continuous map $f: A \to B$ with a continuous inverse.

We say that a complex $\mathcal{C}$ is an *n-disk* if $|\mathcal{C}|$ is homeomorphic to $|\mathcal{S}^n|$, and it is an *$(n-1)$-sphere* if $|\mathcal{C}|$ is homeomorphic to $|\mathcal{S}^{n-1}|$.

2.5. Abstract Simplexes and Complexes. The geometric representations we have given for simplexes and complexes are not always convenient, and we therefore introduce the "complementary" notions of *abstract simplexes* and *abstract complexes*.

Definition 2.16. An *abstract simplex* S is a finite, nonempty set.

The *dimension* of S is one less than its cardinality. Each nonempty subset T of S is called a *face* of S. Each element of S is called a *vertex* of S. Geometric and abstract simplexes are closely related: any affinely independent set of vectors $\{\vec{v}_0, \ldots, \vec{v}_n\}$ span both a geometric and abstract simplex.

Definition 2.17. An *abstract complex* $\mathcal{K}$ is a collection of abstract simplexes closed under containment, that is, if S is in $\mathcal{K}$, so is any face of S.

The notions of *dimension, join*, and *subcomplex* are defined for abstract simplexes and complexes in the obvious way.

Definition 2.18. Let $\mathcal{G}$ be a geometric complex, and let V be the vertex set of $\mathcal{G}$. If $\mathcal{A}$ is the abstract complex of all subsets S of V such that S spans a simplex in $\mathcal{G}$, then $\mathcal{A}$ is called the *vertex scheme* of $\mathcal{G}$.

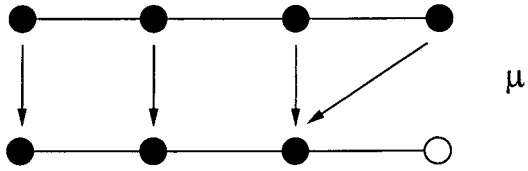

FIG. 5. A simplicial map.

Definition 2.19. Two abstract complexes $\mathcal{K}$ and $\mathcal{L}$ are *isomorphic* if there is a bijective correspondence ψ between their vertex sets such that a set S of vertexes is in $\mathcal{K}$ if and only if $\psi(S) \in \mathcal{L}$. The bijective correspondence ψ is called an *isomorphism*.

The proof of the following theorem can be found in most standard textbooks on algebraic topology [Munkres 1984; Spanier 1966].

THEOREM 2.20. *Every abstract complex $\mathcal{A}$ is isomorphic to the vertex scheme of some geometric complex $\mathcal{G}$ in* $\mathbf{R}^{2\,dim(\mathcal{A})+1}$.

2.6. SIMPLICIAL MAPS AND SUBDIVISIONS. In the rest of this paper, it is convenient to use abstract and geometric simplexes and complexes more or less interchangeably. The remainder of this section, however, focuses on geometric complexes. We first define the notions of vertex maps and simplicial maps.

Definition 2.21. Let $\mathcal{K}$ and $\mathcal{L}$ be complexes, possibly of different dimensions. A *vertex map* μ: $skel^0(\mathcal{K}) \to skel^0(\mathcal{L})$ carries vertexes of $\mathcal{K}$ to vertexes of $\mathcal{L}$. The map is a *simplicial map* if it also carries simplexes of $\mathcal{K}$ to simplexes of $\mathcal{L}$. Any simplicial map μ induces a piece-wise linear map $|\mu|$: $|\mathcal{K}| \to |\mathcal{L}|$ as follows. Every point $\vec{k}$ of $|\mathcal{K}|$ has a unique representation as

$$\vec{k} = \sum k_i \cdot \vec{k}_i,$$

where the $\vec{k}_i$ span a simplex in $\mathcal{K}$, $0 < k_i \leq 1$, and $\Sigma\, k_i = 1$. The k_i are called the *barycentric coordinates* of $\vec{k}$. Define

$$|\mu|(\vec{k}) = \sum k_i \cdot \mu(\vec{k}_i)$$

Henceforth, unless stated otherwise, all maps between complexes are assumed to be simplicial. An example of a simplicial map is given in Figure 5.

Definition 2.22. Let $\mathcal{A} \subset \mathcal{B}$, and ϕ: $\mathcal{A} \to \mathcal{C}$. A simplicial map ψ: $\mathcal{B} \to \mathcal{C}$ *extends* ϕ if they agree on $\mathcal{A}$.

We note that a simplex and its image under a simplicial map need not have the same dimension. As seen in Figure 5, the simplicial map may "collapse" some simplexes.

Definition 2.23. A simplicial map ϕ: $\mathcal{B} \to \mathcal{C}$ *collapses* a simplex T^m, $m > 0$, if $\dim(\phi(T^m)) = 0$.

A simplicial map μ: $\mathcal{K} \to \mathcal{L}$ is *noncollapsing* if it preserves dimension, that is, for all $S \in \mathcal{K}$: $\dim(\mu(S)) = \dim(S)$.

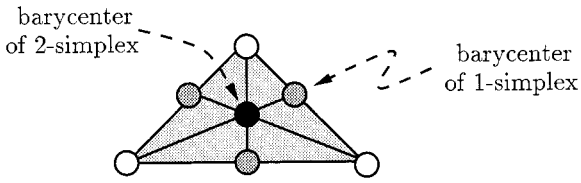

FIG. 6. A barycentric subdivision of a 2-simplex.

Definition 2.24. A *coloring* of an n-dimensional complex $\mathcal{H}$ is a noncollapsing simplicial map $\chi: \mathcal{H} \to \mathcal{S}$, where S is an n-simplex.

Intuitively, a coloring corresponds to a labeling of the vertexes of the complex such that no two neighboring vertexes (vertexes connected by a 1-simplex) have the same label. A *chromatic complex* or *colored complex* $(\mathcal{H}, \chi)$ is a complex $\mathcal{H}$ together with a coloring χ of $\mathcal{H}$. An example of a chromatic complex is given in Figure 33, where the colors are the letters $\{P, Q, R\}$. When it is clear from the context, we specify the chromatic complex $(\mathcal{H}, \chi)$ simply as the complex $\mathcal{H}$, omitting explicit mention of the coloring χ.

Definition 2.25. Let $(\mathcal{H}, \chi_{\mathcal{H}})$ and $(\mathcal{L}, \chi_{\mathcal{L}})$ be chromatic complexes, and let $\mu: \mathcal{H} \to \mathcal{L}$ be a simplicial map. We say that μ is *color-preserving* (or *chromatic*) if, for every vertex $\vec{v} \in \mathcal{H}$, $\chi_{\mathcal{H}}(\vec{v}) = \chi_{\mathcal{L}}(\mu(\vec{v}))$.

In other words, μ is color-preserving if it maps each vertex in $\mathcal{H}$ to a vertex in $\mathcal{L}$ of the same color. Except when otherwise noted, all simplicial maps considered in this paper are color-preserving.

Definition 2.26. Let $\mathcal{H}$ be a complex in $\mathbf{R}^{\ell}$. A complex $\sigma(\mathcal{H})$ is a *subdivision* of $\mathcal{H}$ if:

—*Each simplex in* $\sigma(\mathcal{H})$ *is contained in a simplex in* $\mathcal{H}$.

—*Each simplex of* $\mathcal{H}$ *is the union of finitely many simplexes in* $\sigma(\mathcal{H})$.

One type of subdivision of particular interest in our proof is the barycentric subdivision from classical algebraic topology [Munkres 1984]. Let $S^n = \{\vec{s}_0, \ldots, \vec{s}_n\}$ be an n-simplex of some complex $\mathcal{H}$. The point

$$\vec{b} = \sum_{i=0}^{n} \left(\frac{\vec{s}_i}{n+1} \right)$$

is the *barycenter* of S^n. In particular, if S^n is a vertex, then $\vec{b} = S^n$.

Definition 2.27. The *barycentric subdivision* of a complex $\mathcal{H}$, denoted $\beta(\mathcal{H})$ is defined as follows. Its vertexes are the barycenters of the simplexes of $\mathcal{H}$. For each ordered sequence $S_0, \ldots, S_m$ of simplexes of $\mathcal{H}$ where S_i is a face of S_{i+1} $(i = 0, \ldots, m-1)$, the sequence of corresponding barycenters is the set of vertexes of a simplex of $\beta(\mathcal{H})$. Only simplexes obtained in this manner are in $\beta(\mathcal{H})$.

Figure 6 shows $\beta(\mathcal{S})$ defined on a 2-complex $\mathcal{S}$.

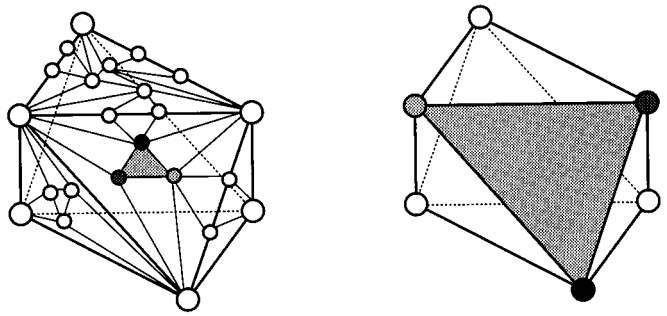

FIG. 7. A simplex and its carrier.

Definition 2.28. If S is a simplex of $\sigma(\mathcal{K})$ or a point in $|\mathcal{K}|$, the *carrier* of S, denoted carrier$(S, \mathcal{K})$ is the unique smallest $T \in \mathcal{K}$ such that $S \subset |T|$.

Figure 7 illustrates the notions of subdivisions and carriers. It shows a complex on the right, and a subdivision on the left, and highlights a simplex S in the subdivision together with its carrier.

One drawback of the barycentric subdivision is that a barycentric subdivision of a chromatic complex is typically not chromatic.

Definition 2.29. A *chromatic subdivision* of $(\mathcal{K}, \chi_{\mathcal{K}})$ is a chromatic complex $(\sigma(\mathcal{K}), \chi_{\sigma(\mathcal{K})})$ such that $\sigma(\mathcal{K})$ is a subdivision of $\mathcal{K}$, and for all S in $\sigma(\mathcal{K})$, $\chi_{\sigma(\mathcal{K})}(S) \subseteq \chi_{\mathcal{K}}(\text{carrier}(S, \mathcal{K}))$.

Figure 33 shows a chromatic subdivision of a complex $\mathcal{S}$ defined on a 2-simplex S. We call this specific type of subdivision the *standard chromatic subdivision* and will use it extensively later in the paper. All subdivisions we consider will be chromatic unless, as in the case of the barycentric subdivision, we explicitly state otherwise.

Definition 2.30. A simplicial map $\mu: \sigma_1(\mathcal{K}) \to \sigma_2(\mathcal{K})$ between chromatic subdivisions of $\mathcal{K}$ is *carrier-preserving* if for all $S \in \sigma_1(\mathcal{K})$, carrier$(\mu(S), \mathcal{K}) \subseteq$ carrier$(S, \mathcal{K})$.

2.7. SIMPLICIAL COMPLEXES AND TASKS. Earlier in this section, we defined the notion of a decision task in terms of input and output vectors. That definition was intended to help the reader understand what a decision task is, but it lacks the mathematical structure necessary to prove interesting results. We now reformulate this definition in terms of simplicial complexes. We present a topological specification that replaces the vector-based task specification of Section 2.1. A formal proof of the correspondence among the two representations is beyond the scope of this paper and can be found in Hoest [1997].

We will construct the topological specification using abstract simplexes and complexes. The reader should note that it follows from Theorem 2.20 that there exists a representation using geometric simplexes and complexes, for which the vertex scheme is isomorphic to the abstract representation. To illustrate our constructions, we accompany the formal definitions with examples of how they are used to represent the following variant[4] of the well-known Renaming decision task first introduced and studied by Attiya et al. [1990].

[4] The full renaming problem is treated in the sequel. It assumes an additional symmetry property.

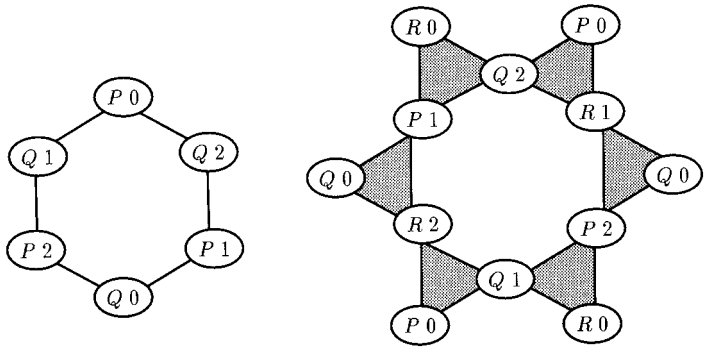

FIG. 8. Some output complexes for the renaming task.

Renaming. The input to each process is a unique *input name* in the range $\{0, \ldots, M - 1\}$. Each participating process chooses a unique *output name* taken from a strictly smaller range $0, \ldots, K$.

Definition 2.31. Let $\vec{I} \in I$ be an input vector. The *input simplex* corresponding to $\vec{I}$, denoted $\mathcal{S}(\vec{I})$, is the abstract colored simplex whose vertexes $\langle P_i, v_i \rangle$ correspond to the participating entries in $\vec{I}$, for which $\vec{I}[i] = v_i \neq \perp$.

The *output simplex* corresponding to $\vec{O}$, denoted $\mathcal{S}(\vec{O})$, is defined accordingly. The process id labeling of a vertex $\vec{v}$ is denoted by $id(\vec{v})$, and the value by $val(\vec{v})$. We use $ids(S)$ to denote the set of process ids of vertexes in a simplex S, and $vals(S)$ to denote the set of values. Figure 4 shows two triangles (2-simplexes) corresponding to two distinct final states for the 3-process renaming task, one in which process P chooses 0, Q chooses 1, and R chooses 2, and another in which P chooses 3, and Q and R choose the same values. Notice that the vertexes of each simplex are colored by the process ids.

This simplicial representation gives a geometric interpretation to the notion of "similar" system states. The vertexes on the common boundary of the two simplexes are local process states that cannot distinguish between the two global states. Unlike graph-theoretic models (e.g., Biran et al. [1988]), simplicial complexes capture in a natural way the notion of the *degree* of similarity between two states: it is the dimension of the intersection of the two n-simplexes.

Since the sets of input vectors we consider are prefix-closed, we can collect input and output vectors into abstract chromatic complexes (i.e., sets of simplexes closed under containment).

Definition 2.32. The *input complex* corresponding to I, denoted $\mathcal{I}$, is the collection of input simplexes $\mathcal{S}(\vec{I})$ corresponding to the input vectors of I. The *output complex* corresponding to O, denoted $\mathcal{O}$, is the collection of output simplexes $\mathcal{S}(\vec{O})$ corresponding to the output vectors of O.

For example, Figure 8 shows the output complexes for renaming with two processes $\{P, Q\}$ using three names $\{0, 1, 2\}$, three processes $\{P, Q, R\}$ with three names $\{0, 1, 2\}$, and three processes using the four names $\{0, 1, 2, 3\}$. The four-name output complex's polyhedron is homeomorphic (topologically equivalent) to a torus. To see why, notice in Figure 9 that the vertexes on edges of the

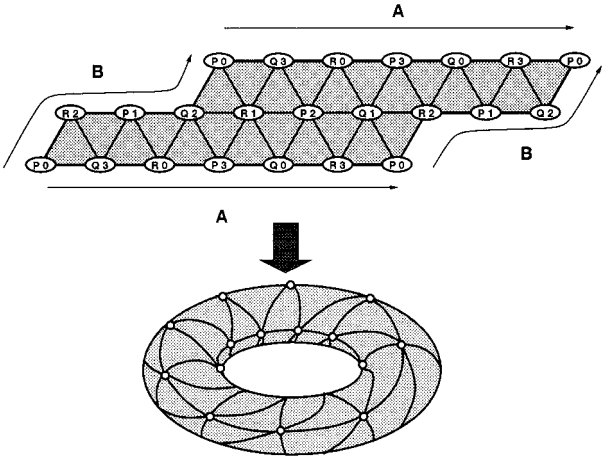

FIG. 9. Output complex for 3-process renaming with 4 names.

complex are the same, so the edges can be "glued together" in the direction of the arrows.

We now construct a topological equivalent of the task specification map $\Delta \subseteq I \times O$.

Definition 2.33. The *topological task specification* corresponding to the task specification Δ, denoted $\Delta \subseteq \mathcal{I} \times \mathcal{O}$, is defined to contain all pairs $(\mathcal{S}(\vec{I}), \mathcal{S}(\vec{O}))$ where $(\vec{I}, \vec{O})$ are in the task specification Δ.

Note that the topological task specification Δ is not a simplicial map.

We can now put these definitions together, as shown schematically in Figure 10.

Definition 2.34. Given an $(n + 1)$-process decision task $\langle I, O, \Delta \rangle$, the corresponding *topological representation* of the task, denoted $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$, consists of an input complex $\mathcal{I}$ corresponding to I, and output complex $\mathcal{O}$ corresponding to O, and a topological task specification Δ corresponding to the task specification Δ.

Usually, we simply refer to a topological task specification as a "task specification."

Definition 2.35. Let U be a set of processes. A *solo* execution by U is one in which all processes in U complete the protocol before any other process takes a step. $\Delta(S^m)$ for $m < n$ is the set of possible outputs of solo executions by the processes in $\text{ids}(S^m)$.

2.8. LINKS, MANIFOLDS, AND CONNECTIVITY. The definitions given so far should suffice to understand the statement (and implications) of our main theorem. The remainder of this section consists of definitions needed to understand the proofs of our theorems, and some of the theorem's applications.

The *star* of a simplex $\mathcal{S} \in \mathcal{C}$, written $st(\mathcal{S}, \mathcal{C})$, is the union of all $|T|$ such that $S \subseteq T$ (see Figure 11). Although stars are defined as polyhedrons, we sometimes treat them as simplicial complexes, relying on context to clarify the precise meaning. The *open star*, written $st^{\circ}(\mathcal{S}, \mathcal{C})$, is the interior of the star. The *link*,

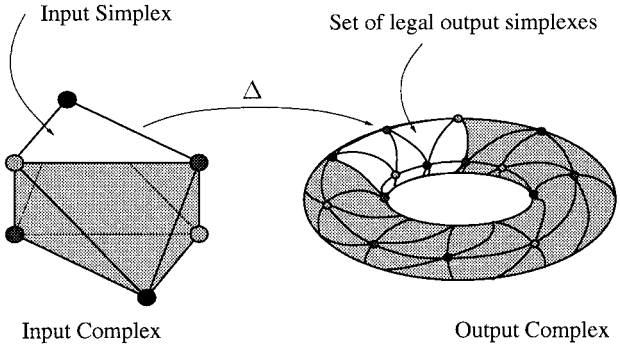

FIG. 10. A decision task.

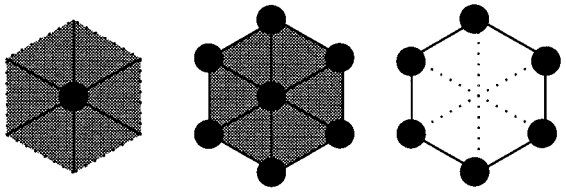

FIG. 11. $st^{\circ}(\vec{v})$, $st(\vec{v})$, and $lk(\vec{v})$.

written $lk(\mathcal{S}, \mathcal{C})$, is the complex consisting of all simplexes in $st(\mathcal{S}, \mathcal{C})$ that contain no vertex of S. These concepts are illustrated in Figure 11. The notion of a link has a simple interpretation. In the renaming complex shown in Figure 8, consider the node labeled $P2$. This node indicates that 2 is a correct output for P. The link of this node is a one-dimensional complex (a hexagon) in which each 1-simplex represents a possible combination of legal outputs for the remaining processes Q and R. In general, in any chromatic complex $\mathcal{C}$, $lk(\mathcal{S}, \mathcal{C})$ is a colored complex with the following interpretation: if we fix the values assigned to processes in $ids(\mathcal{S})$, then $lk(\mathcal{S}, \mathcal{C})$ represents all possible legal ways to assign values to the remaining processes.

A complex $\mathcal{A}$ is an *n-manifold with boundary* if

(1) for every pair of n-simplexes T_0^n, T_1^n in $\mathcal{A}$, there exists a sequence of simplexes $S_0^n, \ldots, S_\ell^n$ such that $T_0^n = S_0^n$, $T_1^n = S_\ell^n$, and $S_i^n \cap S_{i+1}^n$ is an $(n-1)$-simplex, and

(2) every $(n-1)$-simplex is contained in either one or two n-simplexes.

The *boundary complex* of a manifold with boundary is the subcomplex of $(n-1)$-simplexes contained in exactly one n-simplex. If the boundary complex is empty, we refer to the complex simply as a *manifold*.

Some, but not all, the complexes we consider are manifolds. Manifolds satisfy the following property [Glaser 1970; Theorem II.2]:

LEMMA 2.36. *If $\mathcal{M}$ is an n-manifold with boundary, and T^m an interior simplex, then $lk(T^m, \mathcal{M})$ is an $(n-m-1)$-sphere.*

Many complexes of interest have a simple but important topological property: they have no "holes" in certain dimensions. There are several ways to formalize this notion, but the following is the most convenient for our purposes.

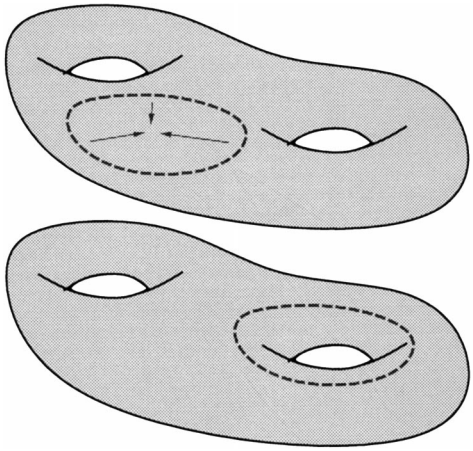

FIG. 12. Contractible and non-contractible loops.

Definition 2.37. For $k > 0$, a nonempty complex $\mathcal{C}$ is *k-connected* [Spanier 1966, p. 51] if, for $m \leq k$, any continuous map of the m-sphere into $|\mathcal{C}|$ can be extended to a continuous map over the $(m + 1)$-disk. It is convenient to define a complex to be (-1)-*connected* if it is nonempty, and any complex to be k-connected for $k < -1$.

A 0-connected complex is usually called *connected*: there is a path linking every pair of vertices. A 1-connected complex is usually called *simply connected*: any loop (closed path) can be continuously deformed to a point.

To illustrate how this formal definition captures the informal notion of a "hole" in a complex, Figure 12 shows a complex (rendered as a surface for simplicity) together with images of a 1-sphere (circle) under continuous maps f and g. The image under f can be contracted to a point, while the image under g circumnavigates a "hole" and cannot be contracted. We will be concerned with proving that certain complexes are k-connected. Although the definition of k-connectivity is topological in nature, we can reason about connectivity in a purely combinatorial way, using the following elementary theorem, proved in the appendix.

THEOREM 2.38. *If $\mathcal{K}$ and $\mathcal{L}$ are complexes such that $\mathcal{K}$ and $\mathcal{L}$ are k-connected, and $\mathcal{K} \cap \mathcal{L}$ is $(k - 1)$-connected, then $\mathcal{K} \cup \mathcal{L}$ is k-connected.*

Before continuing, we note some examples of useful k-connected complexes.

LEMMA 2.39. *The following complexes are k-connected*: (1) *the complex $\mathcal{S}^k$ consisting of a k-simplex and its faces, and* (2) *a cone over an arbitrary $(k - 1)$-dimensional complex.*

This completes the topological concepts necessary to prove our main theorem.

3. *The Main Theorem*

We are now ready to state our main theorem.

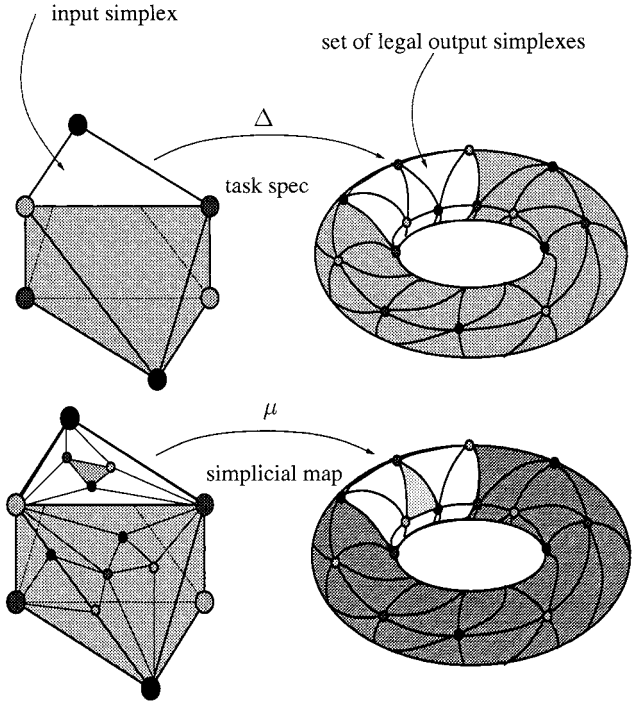

FIG. 13. Asynchronous computability theorem.

THEOREM 3.1 (ASYNCHRONOUS COMPUTABILITY THEOREM). *A decision task* $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ *has a wait-free protocol using read-write memory if and only if there exists a chromatic subdivision* σ *of* $\mathcal{I}$ *and a color-preserving simplicial map*

$$\mu\colon \sigma(\mathcal{I}) \rightarrow \mathcal{O}$$

such that for each simplex S *in* $\sigma(\mathcal{I})$, $\mu(S) \in \Delta(carrier(S, \mathcal{I}))$.

This theorem establishes that task solvability can be characterized in terms of purely topological properties of the task specification, without explicit mention of protocols and executions. A task is solvable if and only if one can subdivide the input complex and map that subdivided complex to the outputs in a way that agrees with Δ. In one direction, we will see that any protocol induces a subdivision, and the structure of that subdivision reflects in a natural way the structure of the protocol.

The theorem is depicted schematically in Figure 13. The figure's top half shows how the task specification Δ takes input simplexes to allowed output simplexes. The bottom half shows how μ maps the subdivided input complex to the output complex in a way consistent with Δ.

In Figure 14, we give a simple example illustrating how the subdivision mentioned in the theorem reflects the unfolding of a protocol execution. Consider the following 2-process task $(\mathcal{I}, \mathcal{O}, \Delta)$, solved by a single-round normal-form protocol. P and Q have respective inputs p and q. They share a two-element array, both elements initialized to $\perp$. P writes its value to the first array element, and then scans the array, while Q writes its value to the second

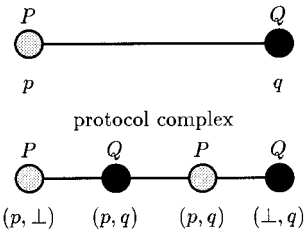

FIG. 14. Simplicial representation of a one-round execution.

element, and then scans the array. Define a process *view* to be the sequence of values it read at the end of the protocol. A one-round, two-process normal-form protocol has only three possible executions: (1) If P reads before Q writes, then P's view is $(p, \perp)$, and Q's is (p, q). (2) If P and Q each reads after the other writes, then both have view (p, q). (3) Q reads before P writes, then P's view is (p, q), and Q's is $(\perp, q)$. These executions define a *protocol complex* $\mathcal{P}$ as follows. There are three 1-simplexes, one for each execution. Each vertex is labeled with a process id and that process's view in that execution. If P reads (p, q), then it cannot "tell" from its view whether P executed before Q or concurrently with Q. The complex $\mathcal{P}$ captures this ambiguity in a geometric way by placing $\langle P, pq\rangle$ in the intersection of the two 1-simplexes representing these two executions. If we identify vertexes $\langle P, p\rangle$ and $\langle Q, q\rangle$ of $\mathcal{I}$ with vertexes $\langle P, p\perp\rangle$ and $\langle Q, \perp q\rangle$ of $\mathcal{P}$, we can see that $\mathcal{P}$ is a subdivision of $\mathcal{I}$, dividing the single edge of $\mathcal{I}$ into three edges.

When a process finishes executing the protocol, it chooses an output value based only on its local state. Formally, this choice is captured by a *decision map* δ carrying vertexes of $\mathcal{P}$ to vertexes of $\mathcal{O}$. Recall that a fixed execution corresponds to a single simplex of $\mathcal{P}$. At the end of this execution, the processes must choose vertexes that lie on a common simplex of $\mathcal{O}$, implying that the vertex map $\delta: \mathcal{P} \to \mathcal{O}$ is a simplicial map. Moreover, because the protocol solves the task, for every simplex S in $\mathcal{P}$, $\delta(S) \in \Delta(\mathit{carrier}(S, \mathcal{I}))$, satisfying the theorem's conditions. In summary, the one-round normal-form protocol defines a protocol complex $\mathcal{P}$ which is the desired subdivision of $\mathcal{I}$, and the decision map defines the desired map to the output complex.

Although this construction of a subdivision and map for a two-process one-round protocol is only an example, the proof of the "only if" part of the theorem is based on the same notions: the protocol itself defines a *protocol complex* that encompasses a subdivision of the input complex having the desired properties.

The proof of the asynchronous computability theorem appears in Section 4 and 5. In the remainder of this section, we give examples of applications of the asynchronous computability theorem.

3.1. BINARY CONSENSUS. Perhaps the simplest decision task is binary *consensus* [Fischer et al. 1985]. As specified in Figure 15, each process starts with a binary input, and eventually chooses a binary output. All output values must agree, and each output must be some process's input.

$\vec{I}$	$\Delta(\vec{I})$	$\vec{I}$	$\Delta(\vec{I})$
$(0, \perp)$	$(0, \perp)$	$(1, \perp)$	$(1, \perp)$
$(\perp, 0)$	$(\perp, 0)$	$(\perp, 1)$	$(\perp, 1)$
$(0, 0)$	$(0, 0)$	$(1, 1)$	$(1, 1)$
$(0, 1)$	$(0, 0), (1, 1)$	$(1, 0)$	$(0, 0), (1, 1)$

FIG. 15. The consensus task.

The input complex for this task is the complex $\mathcal{B}^n$ constructed by assigning independent binary values to $n + 1$ processes. We call this complex the *binary n-sphere* (Figure 16).[5]

The output complex consists of two disjoint n-simplexes, corresponding to decision values 0 and 1. Figure 17 shows the input and output complexes for 2-process binary consensus. In general, the input complex is $(n - 1)$-connected, while the output complex is disconnected. *Consensus* is the generalization of binary consensus to allow input values from an arbitrary range, not only $\{0, 1\}$.

It is well-known that binary consensus has no wait-free read-write protocol [Chor et al. 1987; Herlihy 1991; Loui and Abu-Amara 1987]. Nevertheless, it is instructive to see how this result follows from the asynchronous computability theorem. To keep our presentation simple, we focus on the two-process task.

Processes P and Q are given private binary inputs, and they must agree on one of their inputs. In a solo execution, where P runs alone, it observes only its own input, say 0. Since P must choose a value even if Q never takes a step, P must eventually choose 0. The same is true if Q runs solo with input 1. If, however, P and Q run together, then one of them, say P, must change its tentative decision, while preventing Q from doing the same. At the heart of the published impossibility results for this task is a case analysis of a "bad" execution showing that the commuting and overwriting properties of read and write operations make this kind of synchronization impossible.

The asynchronous computability theorem captures this impossibility in a geometric rather than operational way. Figure 17 shows the input and output complexes for the two-process consensus task. Assume by way of contradiction that a protocol exists. The input complex $\mathcal{I}$ is connected, and so is the subdivision $\sigma(\mathcal{I})$. Simplicial maps preserve connectivity, so $\mu(\sigma(\mathcal{I}))$ is also connected. Let I_{ij} and O_{ij} denote the input and output simplexes where P has value i and Q has value j. Because $\Delta(I_{11}) = O_{11}$, μ carries input vertex $\langle P, 1\rangle$ to output vertex $\langle P, 1\rangle$. Symmetrically, it carries input $\langle Q, 0\rangle$ to output $\langle Q, 0\rangle$. However, these output vertexes lie in distinct connected components of the output complex, so μ cannot be a simplicial map, and by Theorem 3.1, 2-process consensus is not solvable. (Generalizing this argument to n processes yields a simple geometric restatement of the impossibility of wait-free consensus in read/write memory [Chor et al. 1987; Dolev et al. 1987; Loui and Abu-Amara 1987].)

[5] Informally, to see why this complex is homeomorphic to an n-sphere, note that it consists of two subcomplexes: $\mathcal{C}_0^n$ is the set of n-simplexes containing $\langle P_n, 0\rangle$, and $\mathcal{C}_1^n$ the set containing $\langle P_n, 1\rangle$. Each of these is an n-disk, a cone over the binary $(n - 1)$-sphere $\mathcal{B}^{n-1}$. These two n-disks are joined at their boundaries, forming an n-sphere.

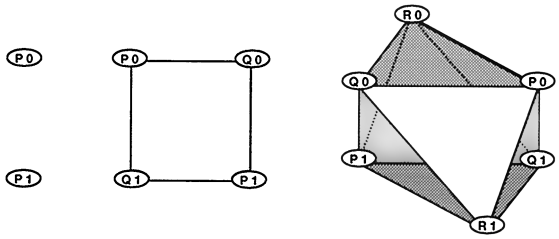

FIG. 16. Binary 0, 1, and 2-spheres.

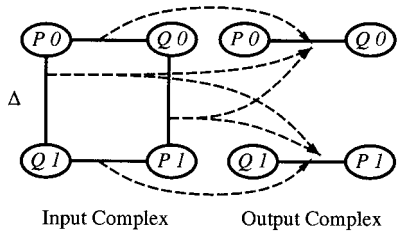

FIG. 17. Simplicial complexes for 2-process consensus.

3.2. QUASI-CONSENSUS. We introduce the following "toy" problem to illustrate further the implications of the theorem. Let us relax the conditions of the consensus task (Figure 18) as follows:

Quasi-Consensus. Each of P and Q is given a binary input. If both have input v, then both must decide v. If they have mixed inputs, then either they agree, or Q may decide 0 and P may decide 1 (but not vice-versa).

Figure 19 shows the input and output complexes for the quasi-consensus task. Is quasi-consensus solvable?

It is easily seen that there is no simplicial map directly from the input complex to the output complex. Just as for consensus, the vertexes of input simplex I_{01} must map to output vertexes $\langle P, 0\rangle$ and $\langle Q, 1\rangle$, but there is no single output simplex containing both vertexes. Nevertheless, there is a map satisfying the conditions of the theorem from a *subdivision* of the input complex. If input simplex I_{01} is subdivided as shown in Figure 20, then it can be "folded" around the output complex, allowing input vertexes $\langle P, 0\rangle$ and $\langle Q, 1\rangle$ to be mapped to their counterparts in the output complex.

Figure 21 shows a simple protocol for quasi-consensus. Recall our earlier explanation that the subdivisions of an input simplex correspond to executions of the protocol. If P has input 0 and Q has input 1, then this protocol admits three distinct executions: one in which both decide 0, one in which both decide 1, and one in which Q decides 0 and P decides 1. These three executions correspond to the three simplexes in the subdivision of I_{01}, which are carried to O_{00}, O_{10}, and O_{11}.

This approach can be extended to tasks involving more than two processes. Recall that in the two-process (one-dimensional) case, the impossibility of consensus follows from the observation that a simplicial map cannot carry a connected component of the subdivided input complex to disconnected components of the output complex. In the $(n + 1)$-process (n-dimensional) case, the

$\vec{I}$	$\Delta(\vec{I})$	$\vec{I}$	$\Delta(\vec{I})$
$(0,\perp)$	$(0,\perp)$	$(1,\perp)$	$(1,\perp)$
$(\perp,0)$	$(\perp,0)$	$(\perp,1)$	$(\perp,1)$
$(0,0)$	$(0,0)$	$(1,1)$	$(1,1)$
$(0,1)$	$(0,0),(1,1),(1,0)$	$(1,0)$	$(0,0),(1,1),(1,0)$

FIG. 18. The quasi-consensus task.

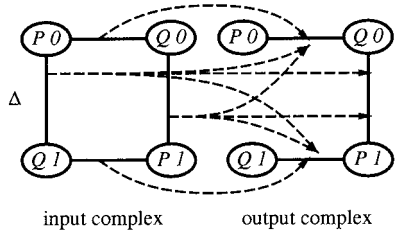

FIG. 19. Input and output complexes for 2-process quasi-consensus.

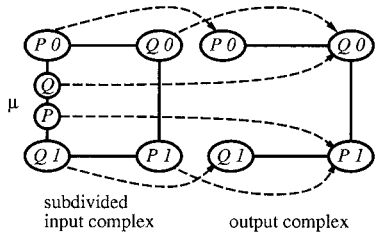

FIG. 20. Subdivided input and output complexes for 2-process quasi-consensus.

```
initially input[P] = nil                initially input[Q] = nil
input[P] := my_input                    input[Q] := my_input
if my_input = 1 then decide 1           if my_input = 0 then decide 0
if input[Q] != 1 then decide 0          if input[P] != 0 then decide 1
decide 1                                decide 0
```

FIG. 21. Quasi-consensus protocols for P and Q.

impossibility of the k-set agreement task follows from a similar observation: a simplicial map cannot carry the boundary of a "solid" disk to the boundary of a "hole."

3.3. SET AGREEMENT. The *k-set agreement* task [Chaudhuri 1990] is a natural generalization of consensus.

***k*-Set Agreement.** Like consensus, each process starts with an input value from some domain, and must choose some process's input as its output. Unlike consensus, all processes together may choose no more than k distinct output values.

Consensus is just 1-set agreement: all processes together may choose no more than $k = 1$ distinct values. When $n = 2$ and $k = 2$, the three processes must return at most two distinct values. As illustrated in Figure 22, the output complex

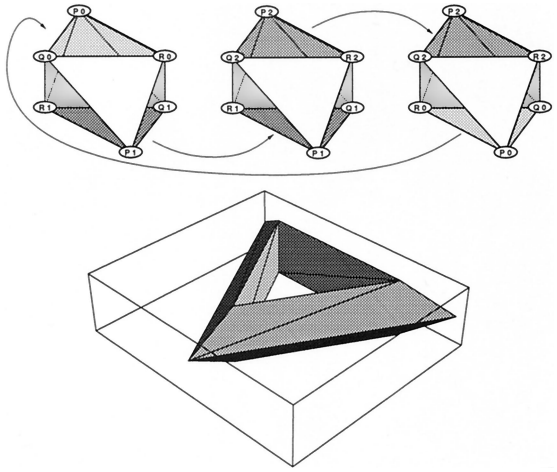

FIG. 22. Output complex for (3,2)-set agreement.

for three-process two-set agreement consists of three binary 2-spheres linked in a ring. This complex is not 1-connected—note the "hole" created by the missing simplexes colored with all three values.

The set agreement problem was first proposed by Chaudhuri [1989], along with a conjecture that it could not be solved when $k \leq n$. This problem remained open until 1993, when three independent research teams, Borowsky and Gafni [1993], Herlihy and Shavit [1993], and Saks and Zaharoglou [1993], proved this conjecture correct.

We now show that the k-set agreement task of Chaudhuri has no wait-free read-write protocol when $k \leq n$. The proof we present is short and rather simple since most of the complexity is hidden in the use of the asynchronous computability theorem. Our proof uses *Sperner's Lemma* [Lefschetz, 1949, Lemma 5.5], a standard tool from algebraic topology.

LEMMA 3.2 (SPERNER'S LEMMA). *Let $\sigma(T)$ be a subdivision of an n-simplex T. If F: $\sigma(T) \rightarrow T$ is a map sending each vertex of $\sigma(T)$ to a vertex in its carrier, then there is at least one n-simplex $S = (\vec{s}_0, \ldots, \vec{s}_n)$ in $\sigma(T)$ such that the $F(\vec{s}_i)$ are distinct.*

We begin with an informal sketch of the three-process case, where $k = 2$, and input and output values taken from {0, 1, 2}. Figure 22 shows the output complex.

Figure 23 shows a subcomplex of the input complex consisting of a simplex T with vertexes $\langle P, 0\rangle$, $\langle Q, 1\rangle$, and $\langle R, 2\rangle$ ($P0$, $P1$, and $R2$ for short), and a collection of 2-simplexes that intersect it along its proper faces. In simplex S_0, all processes have input value 0, and similarly for S_1 and S_2. In simplex S_{01}, all processes have input value 0 or 1, and similarly for S_{12} and S_{02}.

Now assume by way of contradiction that a protocol exists. Recall that the task specification requires each process to decide some process's input. By the asynchronous computability theorem, there exist subdivision σ and color-preserving simplicial map μ consistent with the task specification.

For S_0, the task specification requires each process to decide 0, so for every vertex of $\vec{s}$, $val(\mu(\vec{s})) = 0$. As a result, μ sends the vertex $P0$ of T to an output

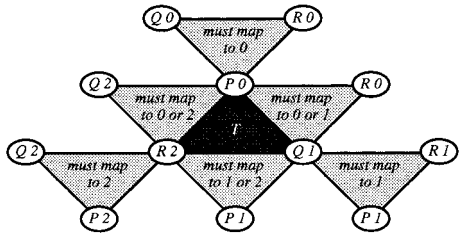

FIG. 23. Part of the set agreement input complex.

vertex also labeled with 0. Similarly, $\mu(Q1)$ and $\mu(R2)$ are respectively labeled with 1 and 2.

For S_{01}, the task specification requires each process to decide 0 or 1, so for every vertex of $\vec{s}$, $val(\mu(\vec{s})) \in \{0, 1\}$. As a result, μ sends every vertex in the subdivided edge $\sigma(P0, Q1)$ to an output vertex labeled with 0 or 1. Similarly, the vertexes of $\mu(\sigma(Q1, R2))$ and $\mu(\sigma(P0, R2))$ are respectively labeled with values from $\{1, 2\}$ and $\{0, 2\}$.

As a result, μ carries each vertex in each subdivided proper face of T to a value in its carrier's set of inputs, as depicted in Figure 24. The map μ thus satisfies the preconditions of Sperner's Lemma, and therefore it carries some 2-simplex in the subdivision $\sigma(T)$ to an output simplex labeled with all three values. The output complex, however, contains no such three-colored simplex, because there is no execution in which three distinct output values are chosen. (Even less formally, it maps to the "hole" in the output complex.)

Here is the full proof.

THEOREM 3.3. *The k-set agreement task has no wait-free read-write protocol for* $k \leq n$.

PROOF. It suffices to prove that there is no solution for $k = n$. Assume by way of contradiction that there is such a protocol. From the k-set agreement task specification, there is an input n-simplex T^n in $\mathcal{I}^n$ with $n + 1$ distinct inputs ($|vals(T^n)| = n + 1$). For every proper face $T^m \subset T^n$, there exists an input simplex $S^n \subset \mathcal{I}^n$ such that $T^m \subset S^n$ and $vals(T^m) = vals(S^n)$. For example, if T^m is a single vertex $\langle P, 1\rangle$, then S^n is the input simplex with $vals(S^n) = \{1\}$. By Theorem 3.1, there exists a color-preserving simplicial map μ: $\sigma(\mathcal{I}^n) \to \mathcal{O}^n$, and by definition, $\mu(\sigma(\mathcal{T}^m))$ must be consistent with $\Delta(S^n)$ for any $S^n \subseteq \mathcal{I}^n$ containing T^m. By the task specification $vals(\Delta(S^n) \subseteq vals(S^n)$ and it follows that the simplicial map μ carries every vertex v of $\sigma(T^m)$ to a vertex in its carrier, hence by Lemma 3.2, there exists a simplex in $\sigma(T^n)$ whose vertexes are mapped to $n + 1$ distinct inputs, that is, to a simplex in $\mathcal{O}$ with $n + 1$ distinct values, a contradiction. □

4. *Necessity*

In this section, we show that the conditions of our theorem are necessary: any decision task $\langle \mathcal{I}, \mathcal{O}, \Delta\rangle$ has a wait-free protocol using read/write memory *only if* there exists a chromatic subdivision $\sigma(\mathcal{I})$ and a color-preserving simplicial map

$$\mu\colon \sigma(\mathcal{I}) \to \mathcal{O}$$

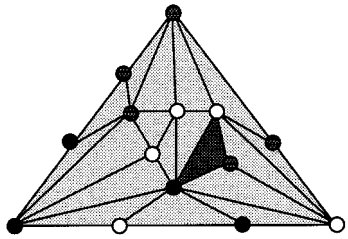

FIG. 24. Sperner's Lemma: At least one simplex has all colors.

such that for each simplex S in $\sigma(\mathcal{I})$, $\mu(S) \in \Delta(carrier(S, \mathcal{I}))$.

In our informal discussion of Figure 14, we represented the three possible executions of a one-round normal form protocol as a simplicial complex which we called the "protocol complex." We observed that this protocol complex induces a subdivision of an input simplex, and that the decision map δ from the protocol complex to the output complex is a simplicial map satisfying the conditions of the theorem.

Our lower-bound proof is just a formalization of the same argument. We first define the protocol complex for a normal-form protocol with multiple processes and multiple rounds. We then show that the protocol complex encompasses a subdivided image of the input complex, and that the decision map induces the desired simplicial map.

First, a note about notational conventions. Some of the results presented here concern properties of arbitrary complexes, while others concern properties of complexes that arise in the context of asynchronous computation. To highlight this distinction, we use symbols such as p and q for dimensions of arbitrary simplexes, while n, as usual, is one less than the number of processes, and m typically ranges between 0 and n.

4.1. PROTOCOL COMPLEXES. At each step in a protocol, the local state of a process consists of its input value together with the sequences of values it scanned. The protocol's global state is just the set of local states, together with the state of the shared atomic snapshot memory $a[0..n]$. It is useful to treat any protocol as an "uninterpreted" protocol in which each process's decision value is just its final local state (bypassing the decision map δ).

We model protocols just like decision tasks. The inputs and outputs for any execution of a protocol $\mathcal{P}$ are given by sets of $(n + 1)$-process input and output vectors, respectively denoted by I and O. As noted in Section 2, because the protocols solve decision tasks, the set I of possible input vectors is prefix-closed. For any protocol, the set O of possible output vectors from all executions of the protocol must also be prefix-closed, for the following reason. Let $\vec{O}$ be an output vector produced by an execution of the protocol, and $\vec{P}$ any prefix. For each i such that $\vec{O}[i] \neq \perp$ and $\vec{P}[i] = \perp$, we can create a new execution in which process i fails just before the FINISH event, that is, just before deciding. Clearly, this execution is a possible execution of $\mathcal{P}$ with output vector $\vec{P}$.

Because the sets of input and output vectors I and O associated with a protocol $\mathcal{P}$ are prefix-closed, there exist corresponding input and output complexes, respectively denoted $\mathcal{I}$ and $\mathcal{P}(\mathcal{I})$.

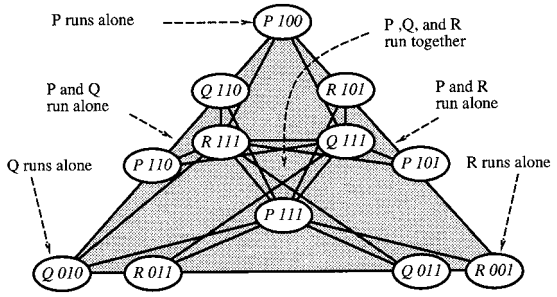

FIG. 25. A one-round protocol complex.

Definition 4.1. The complex $\mathcal{P}(\mathcal{I})$ is called a *protocol complex* over $\mathcal{I}$. Similarly, for a subcomplex $\mathcal{C}$ of the input complex $\mathcal{I}$, $\mathcal{P}(\mathcal{C})$ denotes the set of possible outputs when the protocol is given inputs corresponding to $\mathcal{C}$.

An important special case occurs when $\mathcal{C}$ is S^m, where $0 \leq m \leq n$. The complex $\mathcal{P}(S^m)$ is the set of output simplexes when the processes in $ids(S^m)$ start with their corresponding values in $vals(S^m)$.

The protocol complex satisfies some useful functorial properties, which follow immediately from the definitions. Let $\mathcal{C}_1, \ldots, \mathcal{C}_k$ be subcomplexes of $\mathcal{I}$.

LEMMA 4.2. $\mathcal{P}(\bigcap_{i=0}^{k} \mathcal{C}_i) = \bigcap_{i=0}^{k} \mathcal{P}(\mathcal{C}_i)$.

LEMMA 4.3. $\mathcal{P}(\bigcup_{i=0}^{k} \mathcal{C}_i) = \bigcup_{i=0}^{k} \mathcal{P}(\mathcal{C}_i)$.

What does it mean for a protocol to solve a decision task? Recall that a process chooses a decision value by applying a decision map δ to its local state. We reformulate our main theorem to say that a protocol $\mathcal{P}$ solves a decision task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ if and only if there exists a simplicial, color-preserving *decision map*

$$\delta\colon \mathcal{P}(\mathcal{I}) \rightarrow \mathcal{O},$$

such that for every simplex $S^m \in \mathcal{I}$, and every simplex $T^m \in \mathcal{P}(S^m)$, where $0 \leq m \leq n$, $\delta(T^m) \in \Delta(S^m)$. This definition is just a formal way of stating that every execution of the protocol must yield an output value assignment permitted by the decision problem specification. Though this might seem like a roundabout formulation, it has an important and useful advantage: we have moved from an operational notion of a decision task, expressed in terms of computations unfolding in time, to a purely combinatorial description expressed in terms of relations among topological spaces.

4.2. OUR PROOF STRATEGY. Figure 25 shows the protocol complex for a simple one-round wait-free normal-form protocol. The processes share a three-element atomic snapshot memory array with each entry initialized to 0. Each process P, Q, and R writes 1 to its entry in the array, scans the array's values, and halts. This complex has a simple inductive structure. The vertex at the top "corner" represents a solo execution by P: it writes 1, scans the array, and observes only its own value. The vertexes along the left-hand edge represent solo executions by P and Q together, as in Figure 14. The three vertexes in the interior of the complex represent executions in which all processes' operations are interleaved: each process observes each of the others' values.

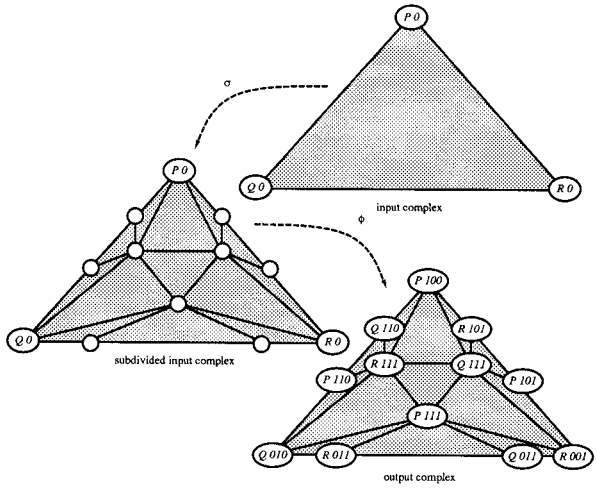

FIG. 26. A span.

Our proof strategy is as follows. For each input simplex S^m, where $0 \leq m \leq n$, we identify a subdivision $\sigma(S^m)$ with a subcomplex of the protocol complex $\mathcal{P}(S^m)$, and construct the simplicial map μ in terms of the decision map δ. The construction is based on the following notion:

Definition 4.4. A *span* for a protocol complex $\mathcal{P}(\mathcal{I})$ is a subdivision $\sigma(\mathcal{I})$ and a color-preserving simplicial map ϕ: $\sigma(\mathcal{I}) \to \mathcal{P}(\mathcal{I})$, such that for every simplex $S \in \sigma(\mathcal{I})$,

$$\phi(S) \in \mathcal{P}(carrier(S, \sigma(\mathcal{I}))). \tag{1}$$

A span σ is thus a subdivision of the input complex with the property that for each input simplex S^m, $0 \leq m \leq n$, the subdivision $\sigma(S^m)$ is mapped to a subcomplex of the protocol complex $\mathcal{P}(S^m)$. This construction is illustrated in Figure 26. The left-hand side shows a three-process input simplex (oval vertexes) which is subdivided (round vertexes) and mapped to a subcomplex of the protocol complex.

The "only if" direction of our main theorem will follow from showing that "if there is a protocol complex then there is a span." The required subdivision σ is the chromatic subdivision of $\mathcal{I}$ induced by the span, and the simplicial map $\mu(\vec{v})$ is just $\delta(\phi(\vec{v}))$, the composition of the span map and the task decision map.

We need to construct the span because the protocol complex itself is not necessarily a subdivision of the input complex (unlike the simple example presented in Figure 14). For example, the one-round three-process protocol complex of Figure 25 is a not subdivided 2-simplex, although it does contain the subdivided 2-simplex shown on the right-hand side of Figure 26.

We construct a span for a given protocol inductively by dimension, successively extending a span defined over the k-skeleton to the $(k + 1)$-skeleton, as in Figure 31.

Here are the principal steps of our construction.

—We show that there are no topological "obstructions" to extending ϕ from the k-skeleton to the $(k + 1)$-skeleton. Section 4.4 provides this first step, showing that for each input simplex S^m, $0 \leq m \leq n$, $\mathcal{P}(S^m)$ is m-connected.

—To maintain the color-preserving nature of σ and ϕ, we check that ϕ can be extended in a way that does not "collapse" simplexes, that is, it does not map higher-dimensional simplexes to lower-dimensional simplexes.

—The key to showing this "noncollapsing" property is the following *local* topological property: for every input simplex S^m, $0 \leq m \leq n$, the link of any k-simplex in $\mathcal{P}(S^m)$ is $(m - k - 2)$-connected, a property we call *link-connectivity*. We address this issue in Section 4.5.

—We complete the proof in Section 4.6 by using connectivity and link-connectivity properties to show that any protocol has a span.

Both m-connectivity and link-connectivity are topological properties of complexes.

4.3. Basic Lemmas. We begin with some general lemmas about simplicial complexes.

Definition 4.5. Complexes $\mathcal{C}_0, \ldots, \mathcal{C}_q$ *cover* $\mathcal{C}$ if $\mathcal{C} = \bigcup_{i=0}^{q} \mathcal{C}_i$. For any index set U, define $\mathcal{C}_U = \bigcap_{i \in U} \mathcal{C}_i$.

Lemma 4.6. *If* $\mathcal{C}_0, \ldots, \mathcal{C}_q$ *cover* $\mathcal{C}$, *then for any index sets* U *and* V,

$$\mathcal{C}_U \cap \mathcal{C}_V = \mathcal{C}_{U \cup V}.$$

Proof. $\mathcal{C}_U \cap \mathcal{C}_V = (\bigcap_{i \in U} \mathcal{C}_i) \cap (\bigcap_{i \in V} \mathcal{C}_i) = \bigcap_{i \in U \cup V} \mathcal{C}_i = \mathcal{C}_{U \cup V}$. □

We will need the following inductive generalization of Theorem 2.38.

Lemma 4.7. *Let* $\mathcal{C}_0, \ldots, \mathcal{C}_q$ *cover* $\mathcal{C}$, *and* $k > 0$ *be such that for all* U, $\mathcal{C}_U$ *is* $(k - |U|)$*-connected. If* $U_0, \ldots, U_\ell$ *are index sets of a given size* u, $\ell + u \leq q + 1$, *such that for each distinct* i *and* j, $\{i\} = U_i - U_j$, *then*

$$\bigcup_{i=0}^{\ell} \mathcal{C}_{U_i} \text{ is } (k - u)\text{-connected.}$$

Proof. If $k = u - 1$, then this lemma simply states that the union of nonempty complexes is nonempty. Let $k > u - 1$. We argue by induction on ℓ. The base case, when $\ell = 0$, is just the hypothesis. For the induction step, when $\ell > 0$, assume that every

$$\mathcal{A} = \bigcup_{i=0}^{\ell-1} \mathcal{C}_{U_i}$$

is $(k - u)$-connected. By the hypothesis,

$$\mathcal{B} = \mathcal{C}_{U_\ell}$$

is $(k - u)$-connected. Their intersection is

$$\mathcal{A} \cup \cap \mathcal{B} = \left(\bigcup_{i=0}^{\ell-1} \mathcal{C}_{U_i} \right) \cap \mathcal{C}_{U_\ell} = \bigcup_{i=0}^{\ell-1} \mathcal{C}_{U_i \cup U_\ell}.$$

Let $V_i = U_i \cup U_\ell$, for $0 \leq i < \ell$. Notice that each $|V_i| = u + 1$, and for each distinct i and j, $\{i\} = V_i - V_j$.

$$\mathcal{A} \cap \mathcal{B} = \bigcup_{i=0}^{\ell-1} \mathcal{C}_{V_i}.$$

The $\mathcal{C}_{V_i}$ satisfy the conditions of the induction hypothesis, so $\mathcal{A} \cap \mathcal{B}$ is $(k - u - 1)$-connected, and the claim now follows from Theorem 2.38. □

4.4. CONNECTIVITY. We now prove a remarkable property of wait-free read/write protocol complexes: for any input simplex S^m, $0 \leq m \leq n$, the protocol complex $\mathcal{P}(S^m)$ is m-connected. In other words, every protocol complex in this model has no "holes."

4.4.1. *The Reachable Complex*. We start with some definitions capturing the way in which the set of executions starting in any global state define a *reachable complex*.

Definition 4.8. Let S^m be an input simplex, $0 \leq m \leq n$, and let s be a global state reached by executing $\mathcal{P}$ from the initial state given by S^m. A simplex R^m of $\mathcal{P}(S^m)$ is *reachable* from s if there is some execution starting from s in which each process in $\mathrm{ids}(R^m)$ completes the protocol with the local state specified in R^m. The *reachable complex* from s, written $\mathcal{R}(s)$, is the complex generated by reachable simplexes from s.

Notice that the reachable complex from the initial state with participating processes and their inputs given by S^m is just $\mathcal{P}(S^m)$. For brevity, we say a state is reachable from an input simplex S^n if it is reachable from the initial state whose process ids and inputs are given by S^n.

If s is a global state in which not all processes have decided, then processes fall into two categories: (1) a *pending* process is about to execute an operation, and (2) a *decided* process has completed its protocol and halted.

Definition 4.9. For a pending process P_i in state s, define the reachable complex $\mathcal{R}_i(s)$ to be the subcomplex of the protocol complex that is reachable after P_i executes its pending operation.

As i ranges over the pending processes, the $\mathcal{R}_i(s)$ cover $\mathcal{R}(s)$. A *pending index set* is a set of indexes of pending processes. If U is a pending index set, define $\mathcal{R}_U(s) = \bigcap_{i \in U} \mathcal{R}_i(s)$. Lemma 4.6 applies. Informally, each simplex in $\mathcal{R}_U(s)$ corresponds to an execution starting in s in which no process can tell which process in U went first.

4.4.2. *Evolving Connectivity*. We now give an informal example showing how the connectivity of the reachable complex evolves as an execution unfolds. Consider the one-round execution of Figure 14, illustrated in Figure 27.

The input complex consists of the single simplex $S^1 = \{\langle P, p\rangle, \langle Q, q\rangle\}$, and the protocol's initial state is s. The reachable complex $\mathcal{P}(S^1)$ encompasses three 1-simplexes. There are two pending operations in s: an update by P and an update by Q. If P goes first, the reachable complex $\mathcal{R}_P(s)$ encompasses the simplexes $\{\langle P, (p, \perp)\rangle, \langle Q, (p, q)\rangle\}$, $\{\langle P, (p, q)\rangle, \langle Q, (p, q)\rangle\}$, and their faces. The reachable complex $\mathcal{R}_Q(s)$ is defined symmetrically. Let $U = \{P, Q\}$, the set of participating processes. Clearly, $\{\mathcal{R}_i(s) | i \in U\}$ cover $\mathcal{R}(s)$. $\mathcal{R}_U(s) = \mathcal{R}_P(s) \cap \mathcal{R}_Q(s)$ is the single simplex $\{\langle P, (p, q)\rangle, \langle Q, (p, q)\rangle\}$ reached in the execution in which both updates occurred before either scan (since updates

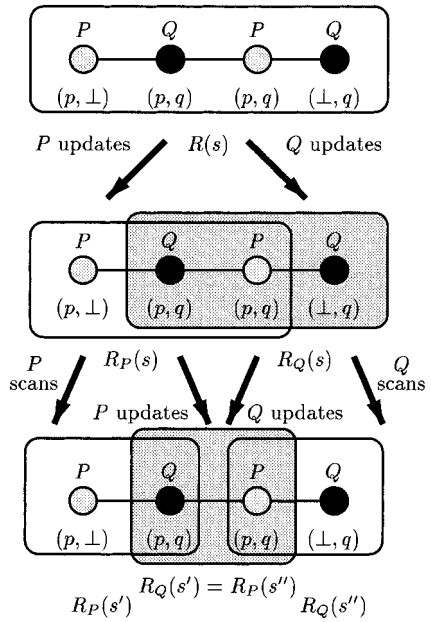

FIG. 27. Connectivity of reachable complexes.

commute, their order does not matter). $\mathcal{R}_U(s)$ is 1-connected. To show that $\mathcal{P}(S^1)$ is 1-connected, Theorem 2.38 implies that it is enough to show that both $\mathcal{R}_P(s)$ and $\mathcal{R}_Q(s)$ are 0-connected (connected in the graph-theoretic sense).

First, let us check that $\mathcal{R}_P(s)$ is connected (a symmetric argument holds for $\mathcal{R}_Q(s)$). Let s' be the global state if P updates $a[P]$ in s, and s'' the global state if Q updates $a[Q]$. There are two pending operations in s': Q's update changing $a[Q]$ to q, and P's scan. If P goes first in s', the reachable complex consists of the 1-simplex $\{\langle P, (p, \perp)\rangle, \langle Q, (p, q)\rangle\}$. If Q goes first, the reachable complex consists of $\{\langle P, (p, q)\rangle, \langle Q, (p, q)\rangle\}$. Their intersection $\mathcal{R}_U(s') = \mathcal{R}_P(s') \cap \mathcal{R}_Q(s')$ cannot contain vertexes of P since in $\mathcal{R}_Q(s')$, Q's update of $a[Q]$ to q must be reflected in the view returned by P's scan. Thus, this intersection is nonempty, containing the 0-simplex $\langle Q, (p, q)\rangle\}$ of Q, which is 0-connected. The 1-connectivity of $\mathcal{R}_P(s)$ follows from Theorem 2.38, since each outcome 1-simplex is connected and their intersection is 0-connected.

4.4.3. *Proof of Connectivity*. We now present the complete proof. Instead of exhaustively considering all executions, as in the example above, we concentrate on a specific "critical" state and argue by contradiction.

Definition 4.10. A global state s is *critical* for a property $\wp$ if $\wp$ does not hold in s, and a step by any pending process will bring the protocol's execution to a state where $\wp$ henceforth holds.

LEMMA 4.11. *If $\wp$ is a property that does not hold in some state s but does hold in the final state of every execution, then $\wp$ has a critical state.*

PROOF. A process is *noncritical* if its next step will not make $\wp$ henceforth hold. Starting from state s, repeatedly pick a noncritical pending process and run it until it is no longer noncritical. Because the protocol must eventually terminate

in a state where $\wp$ holds, advancing noncritical processes in this way will eventually leave the protocol in a state where $\wp$ does not hold, but all processes are either decided or about to make $\wp$ henceforth *true*. This state is the desired critical state. □

We will now show that in any state reachable from any input simplex S^n, $\mathcal{P}(S^n)$ satisfies the conditions of Lemma 4.7. Informally, our proof strategy proceeds by contradiction. Assume the claim is initially false. Since the reachable complex eventually shrinks to a single simplex, it eventually satisfies the desired properties, so by Lemma 4.11 we can run the protocol to a critical state. We then analyze the possible interactions of the pending operations to show that the reachable complex must have satisfied the conditions to begin with, yielding a contradiction. A similar strategy was used by Fischer et al. [1985] to prove the impossibility of asynchronous consensus.

LEMMA 4.12. *For any input simplex S^n, $\mathcal{P}(S^n)$ is n-connected.*

PROOF. By way of contradiction, let $\mathcal{P}$ be an $(n + 1)$-process protocol for which the claim is false. Pick $\mathcal{P}$ so that n is minimal. Let $\wp$ be the property "$\mathcal{R}(s)$ is n-connected." If s is any final state of $\mathcal{P}$, then $\mathcal{R}(s)$ is a single simplex, which is n-connected (Lemma 2.39). Because $\wp$ holds in every final state, Lemma 4.11 implies that $\wp$ has a critical state s.

We claim that for every pending set U, $\mathcal{R}_U(s)$ is $(n - |U| + 1)$-connected. We proceed by a case analysis. Since $\wp$ is *true* in any final state, we can restrict our attention to nonempty pending sets.

Suppose U consists entirely of scans. In every execution leading to a simplex in $\mathcal{R}_U(s)$, each pending scan is ordered before any update. Because scans commute, each such execution is equivalent to one in which all processes in U perform their scans before any other operation occurs. If s' is the state reached from s by executing all pending scans in U, then $\mathcal{R}(s') = \mathcal{R}_U(s)$. Because s is critical, $\mathcal{R}(s') = \mathcal{R}_U(s)$ is n-connected. Because $|U| > 0$, $\mathcal{R}_U(s)$ is $(n - |U| + 1)$-connected.

Suppose U consists entirely of updates. Recall that in normal form protocols, processes update an atomic snapshot memory a, where each new value is distinct from any earlier value. In every execution leading to a simplex in $\mathcal{R}_U(s)$, each pending update must be ordered before any scan. Because updates commute, each such execution is equivalent to one in which all processes in U perform their updates before any other operation occurs. If s' is the state reached from s by executing all pending updates in U, then $\mathcal{R}(s') = \mathcal{R}_U(s)$. Because s is critical, $\mathcal{R}(s') = \mathcal{R}_U(s)$ is n-connected. Because $|U| > 0$, $\mathcal{R}_U(s)$ is $(n - |U| + 1)$-connected.

Finally, suppose both scans and updates appear in U. Let $U = R \cup W$, where R (respectively, W) is the set of processes with pending scans (updates). Suppose $P_i \in R$ is about to scan, and $P_j \in W$ is about to update $a[j]$ from v to v'. In every simplex in $\mathcal{R}_i(s)$, P_i's scan returns v, while in every simplex in $\mathcal{R}_j(s)$, it returns v'. As a result, P_i has no vertexes in $\mathcal{R}_i(s) \cap \mathcal{R}_j(s)$. More generally, $\mathcal{R}_U(s)$ contains no vertex of any process in R. In every execution leading to a simplex in $\mathcal{R}_U(s)$, each update in W is ordered before any scan by a process in $ids(\mathcal{R}_U(s))$. Conversely, any execution from s by processes not in R in which all updates in W precede any other operation is in $\mathcal{R}_U(s)$. Let s' be the state

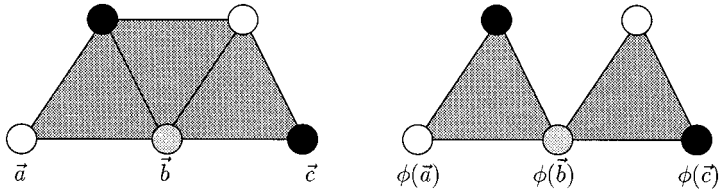

FIG. 28. Complexes $\mathcal{A}$ (left) and $\mathcal{B}$ (right).

reached from s by executing all pending updates in W. Since updates commute, their order is unimportant.

Let $\mathcal{P}'$ be the $(n - |R| + 1)$-process protocol with initial state s' identical to $\mathcal{P}$ except that the processes in R do not participate. Let $\mathcal{R}'(s')$ be the reachable complex for $\mathcal{P}'$ from s'. We have just argued that $\mathcal{R}_U(s) = \mathcal{P}'(s')$. Because $|R| > 0$, and $\mathcal{P}$ was chosen to be minimal, $\mathcal{R}'(s)$ is $(n - |R| + 1)$-connected, and because $|U| > |R|$, it is also $(n - |U| + 1)$-connected.

In all cases, we have shown that $\mathcal{R}_U(s)$ is $(n - |U| + 1)$-connected. By Lemma 4.7, $\mathcal{R}(s) = \bigcup \mathcal{R}_i(s)$ is n-connected. It follows that $\wp$ holds in s, contradicting our assumption that s is a critical state for $\wp$. We have shown that $\mathcal{R}(s)$ is n-connected for every state s. If s is the initial state given by S^n, then $\mathcal{R}(s) = \mathcal{P}(S^n)$ is n-connected for every input simplex S^n. □

For any input simplex S^m, $0 \leq m \leq n$, $\mathcal{P}(S^m)$ can be considered the protocol complex for an $(m + 1)$-process protocol.

COROLLARY 4.13. *For any input simplex S^m, $0 \leq m \leq n$, $\mathcal{P}(S^m)$ is m-connected.*

Note that for any input n-complex $\mathcal{I}$, $\mathcal{P}(\mathcal{I})$ is not necessarily n-connected, since $\mathcal{I}$ itself may not be n-connected.

4.5. LINK CONNECTIVITY. Given Corollary 4.13, it is not hard to construct a subdivision $\sigma(\mathcal{I})$ and a simplicial map μ: $\sigma(\mathcal{I}) \to \mathcal{P}(\mathcal{I})$ satisfying Eq. 1. This construction suffices to show this paper's principal impossibility results (such as the impossibility of k-set agreement), but it is not yet enough to construct an algorithm.

The missing property is that μ must be *color preserving*: for every vertex $\vec{v}$, $id(\mu(\vec{v}))$ must equal $id(\vec{v})$. This property cannot be taken for granted: the existence of a simplicial map does not always guarantee the existence of a color-preserving simplicial map. Consider the following simple example. Let $\mathcal{A}$ and $\mathcal{B}$ be the colored complexes shown in Figure 28, and ϕ the simplicial map carrying vertexes $\vec{a}$, $\vec{b}$ and $\vec{c}$ to $\phi(\vec{a})$, $\phi(\vec{b})$, and $\phi(\vec{c})$, respectively. Does there exist a chromatic subdivision σ of $\mathcal{A}$ and a simplicial map ψ: $\sigma(\mathcal{A}) \to \mathcal{B}$ extending ϕ? It is not difficult to construct a chromatic subdivision σ and a *non* color-preserving simplicial map satisfying these conditions, but it turns out that no color-preserving map is possible. For any subdivision σ, one can show that $lk(\vec{b}, \sigma(\mathcal{A}))$ must be connected. By contrast, $lk(\phi(\vec{b}), \mathcal{B})$ is not connected. Some vertex $\vec{x}$ in $lk(\vec{b}, \sigma(\mathcal{A}))$ must map to $\phi(\vec{a})$, and some $\vec{y}$ to $\phi(\vec{c})$, and the path between $\vec{x}$ and $\vec{y}$ in $lk(\vec{b}, \sigma(\mathcal{A}))$ must map to a path linking the disconnected components of $lk(\phi(\vec{b}), \mathcal{B})$, a contradiction.

To ensure that μ is color preserving, we must prove one more property of each protocol complex $\mathcal{P}(S^m)$, $0 \leq m \leq n$. In addition to being m-connected (Corollary 4.13), $\mathcal{P}(S^m)$ is also *link-connected*.

Definition 4.14. A p-complex $\mathcal{C}$ is *link-connected* if for all simplexes $T^q \in \mathcal{C}$, $0 \leq q \leq p$, $lk(T^q, \mathcal{C})$ is $(p - q - 2)$-connected.

A p-complex can be p-connected without being link-connected, and vice-versa.

LEMMA 4.15. *If $\mathcal{C}$ is link-connected, so is $lk(T, \mathcal{C})$ for any simplex $T \in \mathcal{C}$.*

PROOF. For every simplex S of $\mathcal{C}$ and T of $lk(S, \mathcal{C})$,

$$lk(T, lk(S, \mathcal{C})) = lk(T \cdot S, \mathcal{C}). \qquad \square$$

LEMMA 4.16. *For every input simplex S^n, and simplex $T^m \in \mathcal{P}(S^n)$, $lk(T^m, \mathcal{P}(S^n))$ is $(n - m - 2)$-connected.*

PROOF. The proof resembles the proof of Lemma 4.12.

By way of contradiction, let $\mathcal{P}$ be an $(n + 1)$-process protocol for which the claim is false. Pick $\mathcal{P}$ so that n is minimal. For a global state s, let $\mathcal{R}(s)$ be the reachable complex from s, and $\mathcal{Q}(s) = lk(T^m, \mathcal{R}(s))$ (empty if T^m is not in $\mathcal{R}(s)$). Let $\wp$ be the property

$$T^m \in \mathcal{R}(s) \Rightarrow \mathcal{Q}(s) \text{ is } (n - m - 2)\text{-connected.}$$

Initially, $\wp$ is false by assumption. In every final state s, either T^m is not in $\mathcal{R}(s)$, or $\mathcal{Q}(s)$ is a single $(n - m - 2)$-simplex (which is $(n - m - 2)$-connected). Either way, $\wp$ holds in every final state. By Lemma 4.11, $\wp$ has a critical state s. Notice that because $\wp$ is *false* in s, T^m is in $\mathcal{R}(s)$.

As usual, for each P_i pending in s, $\mathcal{R}_i(s)$ is the reachable complex after P_i executes its operation, and for each set U of pending processes in s, $\mathcal{R}_U(s) = \bigcap_{i \in U} \mathcal{R}_i(s)$. Define $\mathcal{Q}_i(s) = lk(T^m, \mathcal{R}_i(s))$, and $\mathcal{Q}_U(s) = lk(T^m, \mathcal{R}_U(s))$, the $\mathcal{Q}_i(s)$ cover $\mathcal{Q}(s)$, and Lemma 4.6 applies.

Define a pending operation to be *preserving* if it leaves T^m within the reachable complex. There must be a preserving operation pending in s, because otherwise T^m would not be reachable, and $\wp$ would be *true*. Let U be any non-empty set of pending preserving operations in s. For each P_i in U, that process's pending operation is preserving, so T^m is in each $\mathcal{R}_i(s)$, and therefore in $\mathcal{R}_U(s)$. We now show, by case analysis, that any $\mathcal{Q}_U(s)$ is $(n - m - |U| - 1)$-connected.

Suppose U consists entirely of scans. In every execution leading to a simplex in $\mathcal{R}_U(s)$, each pending scan is ordered before any update. Because scans commute, each such execution is equivalent to one in which all processes in U perform their scans before any other operation occurs. If s' is the state reached from s by executing all pending scans in U, then $\mathcal{R}(s') = \mathcal{R}_U(s)$ and $\mathcal{Q}(s') = \mathcal{Q}_U(s)$. Because s is critical, $\wp$ holds in s'. Since T^m is in $\mathcal{R}_U(s) = \mathcal{R}(s')$, $\mathcal{Q}(s') = \mathcal{Q}_U(s)$ is $(n - m - 2)$-connected. Because $|U| > 0$, $\mathcal{Q}_U(s)$ is $(n - m - |U| - 1)$-connected.

Suppose U consists entirely of updates. Recall that in normal form protocols, processes update an atomic snapshot memory a, where each new value is distinct from any earlier value. In every execution leading to a simplex in $\mathcal{R}_U(s)$, each pending update must be ordered before any scan. Because updates commute,

each such execution is equivalent to one in which all processes in U perform their updates before any other operation occurs. If s' is the state reached from s by executing all pending updates in U, then $\mathcal{R}(s') = \mathcal{R}_U(s)$ and $\mathcal{Q}_U(s) = \mathcal{Q}(s')$. Because s is critical, $\mathcal{Q}(s') = \mathcal{Q}_U(s)$ is $(n - m - 2)$-connected, and because $|U| > 0$, $\mathcal{Q}_U(s)$ is $(n - m - |U| - 1)$-connected.

Finally, suppose both scans and updates appear in U. Let $U = R \cup W$, where $R(W)$ is the set of processes with pending preserving scans (updates). Suppose $P_i \in R$ is about to scan, and $P_j \in W$ is about to update $a[j]$ from v to v'. In every simplex in $\mathcal{R}_i(s)$, P_i's scan returns v, while in every simplex in $\mathcal{R}_j(s)$, it returns v'. As a result, P_i has no vertexes in $\mathcal{R}_i(s) \cap \mathcal{R}_j(s)$. More generally, $\mathcal{R}_U(s)$ contains no vertex of any process in R. In every execution leading to a simplex in $\mathcal{R}_U(s)$, each update in W is ordered before any scan by a process in $ids(\mathcal{R}_U(s))$. Conversely, any execution from s by processes not in R in which all updates in W precede any other operation is in $\mathcal{R}_U(s)$. Let s' be the state reached from s by executing all pending updates in W. Since updates commute, their order is unimportant.

Let $\mathcal{P}'$ be the $(n - |R| + 1)$-process protocol with initial state s' identical to $\mathcal{P}$ except that the processes in R do not participate. Let $\mathcal{R}'(s')$ be the reachable complex for $\mathcal{P}'$ from s', and $\mathcal{Q}'(s') = lk(T^m, \mathcal{R}'(s'))$. We have just argued that $\mathcal{R}_U(s) = \mathcal{P}'(s')$, and $\mathcal{Q}_U(s) = \mathcal{Q}'(s')$. Because $|R| > 0$, and $\mathcal{P}$ was chosen to be minimal, $\mathcal{Q}'(s') = \mathcal{Q}_U(s)$ is $(n - m - |R| - 1)$-connected. Because $|U| > |R|$, $\mathcal{Q}_U(s)$ is also $(n - m - |U| - 1)$-connected.

In all cases, we have shown that $\mathcal{Q}_U(s)$ is $(n - m - |U| - 1)$-connected. By Lemma 4.7, $\mathcal{Q}(s) = \bigcup \mathcal{Q}_i(s)$ is $(n - m - 2)$-connected. It follows that $\wp$ holds in s, contradicting our assumption that s is a critical state for $\wp$. We have shown that $\wp$ must hold in every state, and that every $\mathcal{Q}(s)$ is $(n - m - 2)$-connected. In particular, for every input simplex S^n, $\mathcal{P}(S^n)$ is link-connected. □

For any input simplex S^m, $0 \leq m \leq n$, $\mathcal{P}(S^m)$ can be considered the protocol complex for an $(m + 1)$-process protocol.

COROLLARY 4.17. *For any input simplex S^m, $0 \leq m \leq n$, $\mathcal{P}(S^m)$ is link-connected.*

4.6. EVERY PROTOCOL HAS A SPAN. For our inductive construction, we will need to show that any color and carrier-preserving map from a subdivision of the i-skeleton of $\mathcal{I}$ to $\mathcal{P}(\mathcal{I})$ can be extended up one dimension, to a color and carrier-preserving map of the $(i + 1)$-skeleton.

Recall that a simplicial map *collapses* a simplex of non-zero dimension if it maps that simplex to a vertex. A map is *noncollapsing* if it collapses no simplexes. Clearly, color-preserving maps are noncollapsing. Conversely, if a color-preserving map $\sigma(skel^i(\mathcal{A})) \to \mathcal{B}$ has a noncollapsing extension $\sigma(skel^{i+1}(\mathcal{A})) \to \mathcal{B}$, then that extension is also color-preserving. Consequently, we focus on the circumstances under which maps have noncollapsing extensions.

The following lemma appears in Glaser [1970, Theorem IV.2].

LEMMA 4.18. *Let $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ be complexes such that $\mathcal{A} \subset \mathcal{B}$, and f: $|\mathcal{B}| \to |\mathcal{C}|$ a continuous map such that the vertex map induced by f restricted to $|\mathcal{A}|$ is simplicial. There exists a subdivision τ of $\mathcal{B}$ such that $\tau(\mathcal{A}) = \mathcal{A}$, and a simplicial map ϕ: $\tau(\mathcal{B}) \to \mathcal{C}$ extending the restriction of f to $|\mathcal{A}|$.*

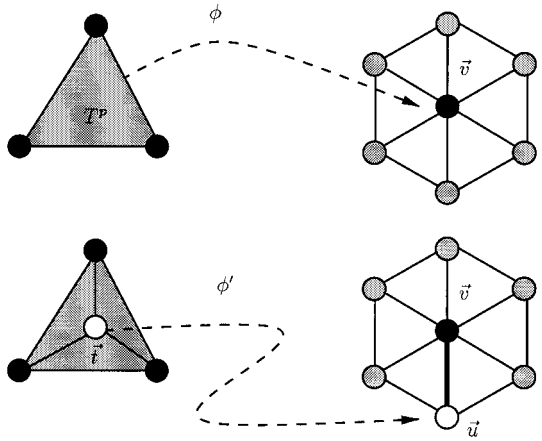

FIG. 29. Eliminating simplex collapse: Top dimension.

Definition 4.19. Let σ be a subdivision of $skel^{p-1}(\mathscr{C})$, for some complex $\mathscr{C}$. The subdivision of $skel^p(S^p)$ obtained by *starring* σ [Munkres 1984, p. 85] is defined as follows. Let $S_0^p, \ldots, S_L^p$ be the p-simplexes of $skel^p(\mathscr{C})$. For $0 \leq i \leq L$, let $\vec{w}_i$ be the barycenter of $|S_i^p|$. Each $\vec{w}_i \cdot \sigma(skel^{p-1}(S_i^p))$ is a subdivision of S_i^p, and the union of these complexes as i ranges from 0 to L is a subdivision of $skel^p(\mathscr{C})$ that agrees with σ on the $(p-1)$-skeleton.

If σ is the trivial subdivision, we can apply this construction to the boundary complex of a single simplex S^p in $\mathscr{C}$, in which case we speak of starring S^p in $\mathscr{C}$.

LEMMA 4.20. *Let $\mathscr{A}$ be a $(p-1)$-sphere, $\mathscr{B}$ a p-disk having $\mathscr{A}$ as boundary, and $\mathscr{C}$ a complex that is $(p-1)$-connected and link-connected. If $\eta: \mathscr{A} \to \mathscr{C}$ is a simplicial map, then*

(1) *there exists a subdivision τ of $\mathscr{B}$ such that $\tau(\mathscr{A}) = \mathscr{A}$,*
(2) *a simplicial map $\phi: \tau(\mathscr{B}) \to \mathscr{C}$ that agrees with η on $\mathscr{A}$, and*
(3) *ϕ collapses no internal simplexes of $\tau(\mathscr{B})$.*

PROOF. Because $\mathscr{C}$ is $(p-1)$-connected, the continuous map $|\eta|$ on $\mathscr{A}$ can be extended to a continuous $f: |\mathscr{B}'| \to |\mathscr{C}|$ whose restriction to $\mathscr{A}$ is simplicial. Conditions 1 and 2 follow immediately from Lemma 4.18.

If Condition 3 does not hold, choose τ and ϕ to minimize (1) the dimension of $\mathscr{C}$, (2) the largest dimension of any collapsed simplex, and (3) the number of collapsing simplexes of that dimension. We will demonstrate a contradiction by "adjusting" τ and ϕ to collapse one fewer simplex of maximal dimension. To adjust ϕ, we subdivide the collapsed simplex T by inserting a new vertex at the barycenter, and then extending ϕ to send that new vertex to a vertex adjacent to $\phi(T)$, resulting in a new subdivision and map that collapses one fewer simplex of maximal dimension.

Suppose ϕ collapses a simplex T^p, where p is the maximal dimension of any simplex in $\mathscr{B}$. As illustrated in Figure 29, where T^p is a triangle, starring T^p yields a new subdivision $\tau'(\mathscr{B})$. Let $\phi(T^p) = \vec{v}$, and let $\vec{t}$ be the barycenter of T^p. If $T_0^{p-1}, \ldots, T_p^{p-1}$ are the $(p-1)$-faces of T^p, then starring T^p (Definition 4.19) yields a new subdivision $\tau'(\mathscr{B})$. Pick $\vec{u}$ such that $(\vec{v}, \vec{u})$ is a 1-simplex in $\mathscr{C}$. Define

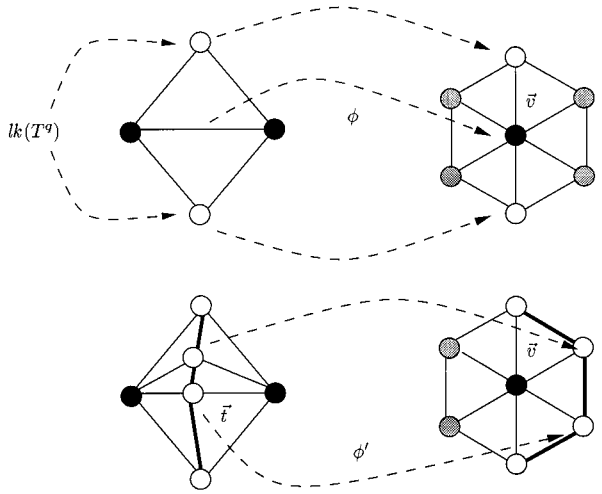

FIG. 30. Eliminating simplex collapse: Intermediate dimensions.

ϕ': $\tau'(\mathcal{B}) \to \mathcal{C}$ such that $\phi'(\vec{t}) = \vec{u}$, and elsewhere $\phi' = \phi$. We have constructed a subdivision τ' and simplicial map ϕ' satisfying Conditions 1 and 2, but collapsing one fewer p-simplex, a contradiction.

Suppose ϕ collapses an internal simplex $T^q \in \tau(\mathcal{B})$ to $\vec{v}$, where $1 \leq q < p$, but collapses no internal simplexes of higher dimension. Our approach is illustrated in Figure 30, where T^q is an edge. Define the subdivision $\tau'(T^q)$ by starring T^q in $\tau(\mathcal{B})$.

Because $q < p$, then $lk(T^q, \mathcal{B})$ is nonempty. The vertexes of any $L^{p-q-1} \in lk(T^q, \mathcal{B})$ are affinely independent of the vertexes of T^q, so $\vec{t}$ is affinely independent of each L^{p-q-1}. Moreover, each $\vec{t} \cdot L^{p-q-1}$ is a simplex, and $\vec{t} \cdot lk(T^q, \mathcal{B})$ is a complex. Because $\mathcal{B}$ is a manifold with boundary, Lemma 2.36 implies that $lk(T^q, \mathcal{B})$ is a $(p - q - 1)$-sphere, and hence $\vec{t} \cdot lk(T^q, \mathcal{B})$ is a $(p - q)$-disk. Because ϕ does not collapse any $(q + 1)$-simplexes, ϕ does not send any vertex of $lk(T^q, \mathcal{B})$ to $\vec{v}$, so ϕ: $lk(T^q, \mathcal{B}) \to lk(\vec{v}, \mathcal{C})$.

We have a simplicial map ϕ carrying the $(p - q - 1)$-sphere $lk(T^q, \mathcal{B})$ to $lk(\vec{v}, \mathcal{C})$, which is $(p - 2)$-connected by Lemma 4.15. Recall that the dimension of $\mathcal{C}$ is the smallest for which Condition 3 fails, and $\dim(lk(\vec{v}, \mathcal{C})) < \dim(\mathcal{C})$, so all three conditions are satisfied: (1) there is a subdivision ρ of $\vec{t} \cdot lk(T^q, \mathcal{B})$, (2) a simplicial map ψ: $\rho(\vec{t} \cdot lk(T^q, \mathcal{B})) \to lk(\vec{v}, \mathcal{C})$ that agrees with ϕ on $lk(T^q, \mathcal{B})$, and (3) ψ collapses no internal simplexes of $\rho(\vec{t} \cdot lk(T^q, \mathcal{B}))$.

The complex $\rho(\vec{t} \cdot lk(T^q, \mathcal{B})) \cdot \mathcal{T}^{q-1}$ is a subdivision of $st(T^q, \mathcal{B})$ that leaves its boundary unchanged. Replacing $st(T^q, \mathcal{B})$ in $\tau(\mathcal{B})$ by this subdivision yields a subdivision $\tau'(\mathcal{B})$. Define ϕ' to agree with ψ on $\rho(\vec{t} \cdot lk(T^q, \mathcal{B}))$, and with ϕ elsewhere. Condition 2 ensures that ϕ and ψ agree on $lk(T^q, \mathcal{B})$, so this map is well-defined. The complex $\tau'(\mathcal{B})$ and map ϕ' satisfy Conditions 1 and 2, but the map collapses one fewer q-simplex, a contradiction. □

LEMMA 4.21. *Let S^p be a p-simplex, $\mathcal{S}^{p-1}$ its boundary complex, and σ a chromatic subdivision of $\mathcal{S}^{p-1}$. Let $\mathcal{C}$ be a p-colored complex that is $(p - 1)$-connected and link-connected, and*

$$\phi\colon \sigma(\dot{\mathcal{S}}^{p-1}) \to \mathcal{C}$$

a color-preserving simplicial map. There exist a chromatic subdivision $\hat{\sigma}$ *of* S^p, *and a color-preserving simplicial map*

$$\hat{\phi}\colon \hat{\sigma}(S^p) \rightarrow \mathcal{C}$$

such that $\hat{\sigma}$ *agrees with* σ *on* $\mathcal{S}^{p-1}$, *and* $\hat{\phi}$ *agrees with* ϕ *on* $\sigma(\mathcal{S}^{p-1})$.

PROOF. The complex $\sigma(\mathcal{S}^{p-1})$ is a $(p-1)$-sphere, and the subdivision of S^p constructed by starring $\sigma(\mathcal{S}^{p-1})$ is a p-disk having $\sigma(\mathcal{S}^{p-1})$ as boundary. By Lemma 4.20, ϕ and σ can be extended to $\hat{\phi}$ and $\hat{\sigma}$ such that $\hat{\sigma}$ agrees with σ on $\mathcal{S}^{p-1}$, and $\hat{\phi}$ is a simplicial map that agrees with ϕ on $\sigma(\mathcal{S}^{p-1})$, and collapses no internal simplexes of $\hat{\sigma}(S^p)$.

It remains to check that $\hat{\phi}$ is color-preserving. Say that T^p is a *boundary simplex* if it can be expressed as $\vec{t} \cdot T^{p-1}$, where T^{p-1} is in $\sigma(\mathcal{S}^{p-1})$. Because ϕ is chromatic on $\sigma(\mathcal{S}^{p-1})$, it carries the face T^{p-1} to a simplex labeled with $(p-1)$ out of the p colors labeling $\mathcal{C}$. Because ϕ does not collapse T^p, it must carry $\vec{t}$ to a vertex labeled with the only remaining color. Therefore, $\hat{\phi}$ is color-preserving on the boundary complex T^p.

Because $\hat{\sigma}(S^p)$ is a p-manifold with boundary, for every simplex T^p in $\hat{\sigma}(S^p)$, there is a sequence of simplexes $S^p_0, \ldots, S^p_\ell$ such that S^p_0 is a boundary simplex, $S^p_\ell = T^p$, and $S^p_i \cap S^p_{i+1}$ is an $(n-1)$-simplex. The claim now follows by inductively extending the same argument along the sequence. □

We now have the tools needed to construct a span.

LEMMA 4.22. *Every wait-free protocol complex has a span.*

PROOF. We build up σ and ϕ by induction on the skeleton of $\mathcal{I}$ as depicted in Figure 31. For each $\vec{v} \in \mathcal{I}$, define $\phi_0(\vec{v}) = \mathcal{P}(\vec{v})$, the unique vertex in the protocol complex corresponding to a solo execution of process $id(\vec{v})$ with input $val(\vec{v})$. This map is color-preserving, and satisfies Eq. (1) of Definition 4.4.

Assume inductively that we have a subdivision σ_{k-1} and a color-preserving simplicial map

$$\phi_{k-1}\colon \sigma_{k-1}(skel^{k-1}(\mathcal{I})) \rightarrow \mathcal{P}(\mathcal{I})$$

satisfying Eq. (1). Let τ be the subdivision of $skel^k(\mathcal{I})$ obtained by starring σ_{k-1}. Let $S^k_0, \ldots, S^k_K$ be all the k-simplexes in $skel^k(\mathcal{I})$. For each S^k_i, $0 \le i \le K$, let $\mathcal{A}_i$ be the complex $\tau(skel^{k-1}(S^k_i))$, and $\mathcal{B}_i$ the complex $\tau(S^k_i)$, and $\mathcal{C}_i$ the complex $\mathcal{P}(S^k_i)$. By Lemma 4.21, there exists a chromatic subdivision τ_i of $\mathcal{B}_i$ such that $\tau_i(\mathcal{A}_i) = \mathcal{A}_i$, a color-preserving simplicial map $\psi_i\colon \tau(\mathcal{B}_i) \rightarrow \mathcal{C}_i$ that agrees with ϕ_{k-1} on the $\mathcal{A}_i$. Because the τ_i agree on $skel^{k-1}(\mathcal{I})$, they induce a subdivision $\sigma_k(skel^k(\mathcal{I}))$. Because the ψ_i also agree on $\sigma_{k-1}(skel^{k-1}(\mathcal{I}))$, they induce a color-preserving simplicial map $\phi_k\colon \sigma_k(skel^k(\mathcal{I})) \rightarrow \mathcal{P}(\mathcal{I})$. Finally, it is immediate from the construction that σ_k and ϕ_k satisfy Eq. (1).

The desired subdivision σ is σ_n, and the desired map ϕ is ϕ_n. (Notice that $skel^n(\mathcal{I}) = \mathcal{I}$.) □

THEOREM 4.23. *If a decision task* $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ *has a wait-free read/write protocol, then there exists a chromatic subdivision* $\sigma(\mathcal{I})$ *and a color-preserving simplicial map* $\mu\colon \sigma(\mathcal{I}) \rightarrow \mathcal{O}$ *such that for each simplex* S^m *in* $\sigma(\mathcal{I})$, $\mu(S) \in \Delta(carrier(S, \mathcal{I}))$.

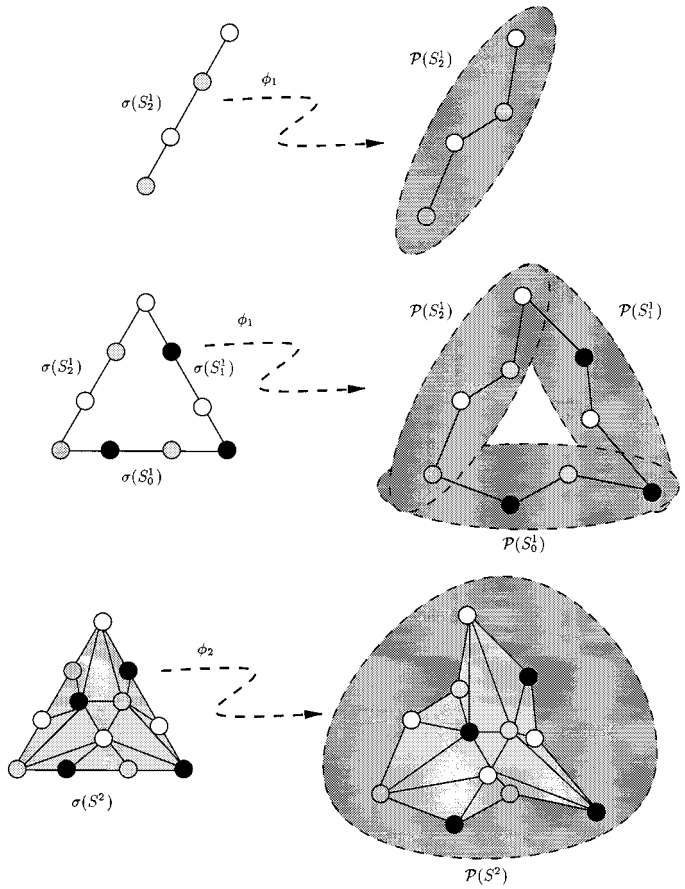

FIG. 31. Inductive span construction.

PROOF. Suppose a protocol exists. By Lemma 4.22, the protocol has a span. Let σ be the chromatic subdivision of $\mathcal{I}$ induced by the span, and let $\mu(S) = \delta(\phi(S))$, the composition of the span map and the decision map. □

5. *Sufficiency*

In this section, we show how to construct an algorithm for any task satisfying the conditions of the asynchronous computability theorem. We now give an overview of this construction.

We introduce the *standard chromatic subdivision* of a complex $\mathcal{I}$, denoted $\chi(\mathcal{I})$, the chromatic analogue of the classical barycentric subdivision. This subdivision is illustrated in Figure 33. We also introduce the *iterated* standard chromatic subdivision $\chi^K(\mathcal{I})$, and the notion of a chromatic subdivision holding a subcomplex fixed.

We will show that we can assume without loss of generality, that the span subdivision $\sigma(\mathcal{I})$ has the form $\chi^K(\mathcal{I})$, for some sufficiently large K. To solve the task, each process P_i begins by placing a token on an input vertex $\vec{s}_i$. As vertexes of $\mathcal{I}$, they are "close", since they span a simplex S. As vertexes of $\chi^K(\mathcal{I})$, however, they are "far apart", since they lie on the boundary of the subcomplex

$\chi^K(S)$. The key insight is that we can construct a protocol for this task by reduction to a variation on *approximate agreement* [Attiya et al. 1990; Fekete 1986], in which the processes start out at the vertexes of S, and after a process of negotiation eventually converge to the vertexes of a single simplex in $\chi^K(S)$. Once P_i has converged on a vertex $\vec{t}_i$ of matching color, it solves the original task by choosing $\mu(\vec{t}_i)$.

We call this process *simplex agreement*. Simplex agreement on $\chi(\mathcal{I})$ is solved by the elegant "participating set" protocol of Borowsky and Gafni [1993]. Simplex agreement on $\chi^K(\mathcal{I})$ is solved by iterating that protocol K times.

Our construction relies on the following theorem, proved below.

Let S^n be a colored n-simplex, χ the standard chromatic subdivision, and σ an arbitrary chromatic subdivision. For sufficiently large K, there exists a color and carrier-preserving simplicial map

$$\phi\colon \chi^K(S^n) \rightarrow \sigma(S^n).$$

This theorem implies that any algorithm for simplex agreement on $\chi^K(S^n)$ yields an algorithm for simplex agreement on an arbitrary $\sigma(S^n)$. Note that this theorem is expressed entirely in terms of combinatorial topology.

5.1. Proof Strategy. To establish the intuition underlying our proof, we give an informal outline of a proof that for sufficiently large K, there exists a carrier-preserving (but not necessarily color-preserving) simplicial map $\phi\colon \chi^K(S^n) \rightarrow \sigma(S^n)$. This claim is a special case of the well-known finite simplicial approximation theorem [Munkres 1984, p. 89].

Here are some useful notions from classical point-set topology.

Definition 5.1. An *open cover for a geometric complex* $\mathcal{C}$ is a finite collection of open sets $U_0, \ldots, U_k$ such that $\mathcal{C} \subseteq \bigcup_{i=0}^{k} U_i$.

The following result is standard [Munkres 1984, p. 89].

Lemma 5.2. *Let $\mathcal{C}$ be a complex,*[6] *and $U_0, \ldots, U_k$ an open cover for $\mathcal{C}$. There exists a $\lambda > 0$ (called a Lebesgue number) such that any set of diameter less than λ lies in a single U_i.*

The open stars around vertexes of $\sigma(S^n)$ form an open cover for $\sigma(S^n)$ with Lebesgue number λ. By choosing K sufficiently large, we can ensure that the diameter of any vertex's star in $\chi^K(S^n)$ is less than λ. It follows that for each vertex $\vec{x}$ in $\chi^K(S^n)$, there is an $\vec{s}$ in $\sigma(S^n)$ such that

$$st(\vec{x}, \chi^K(S^n)) \subset st^\circ(\vec{s}, \sigma(S^n)). \tag{2}$$

Let $\phi(\vec{x}) = \vec{s}$. It is easily shown that ϕ is the desired carrier-preserving simplicial map. Unfortunately, ϕ is not necessarily color-preserving: $id(\vec{x})$ may not equal $id(\phi(\vec{x}))$.

Our goal is to extend Eq. (2) to require that $\vec{x}$ and $\vec{s}$ have matching colors: for each vertex $\vec{x}$ in $\chi^K(S^n)$, there is an $\vec{s}$ in $\sigma(S^n)$ such that

$$st(\vec{x}, \chi^K(S^n)) \subset st^\circ(\vec{s}, \sigma(S^n)) \text{ and } id(\vec{x}) = id(\vec{s}). \tag{3}$$

[6] In fact, $\mathcal{C}$ could be any compact space.

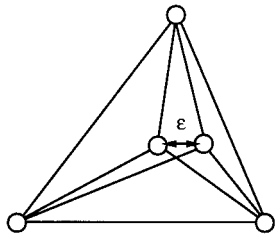

FIG. 32. An ϵ-perturbation.

An immediate difficulty is that $\vec{x}$ itself may not lie in the open star of any $\vec{s}$ of matching color. This situation occurs exactly when $\vec{x}$ lies in $lk(\vec{s}, \sigma(S^n))$, for $\vec{s}$ of matching color. In such a case, however, we can displace $\vec{x}$ by some small distance ϵ within its carrier to bring it within the open star of $\vec{s}$. We refer to such a process as an *ϵ-perturbation* (Figure 32). By applying a suitable ϵ-perturbation, we can ensure that every $\vec{x}$ lies within an open star of matching color.

The *extended star* of a simplex S is the union of the stars of its vertices. For a sufficiently large K, we can ensure that for every n-simplex X^n in $\chi^K(S^n)$, the extended star of X^n lies within some open star $st^\circ(\vec{s}, \sigma(S^n))$. Since the vertexes of X^n are labeled with all n colors, X^n must include one vertex $\vec{x}$ whose color matches that of $\vec{s}$. By construction, $\vec{x}$ and $\vec{s}$ satisfy Eq. 3. The vertex map $\phi(\vec{x}) = \vec{s}$ is color and carrier-preserving, and it is defined on one vertex of each N-simplex of $\chi^K(S^n)$.

The remaining vertexes for which ϕ is *not* defined span an $(n - 1)$-dimensional subcomplex of $\chi^K(S^n)$. We repeat essentially the same construction in a sequence of rounds. In each round, by applying a perturbation and further subdivision, we extend ϕ to one more vertex of each remaining simplex, and the dimension of the subcomplex on which ϕ remains undefined drops by one. After $n + 1$ rounds, we have constructed a color and carrier-preserving vertex map ϕ. Finally, we check that ϕ is simplicial.

Our proof proceeds as follows. In Section 5.2, we define the standard chromatic subdivision. In Section 5.3, we define the simplex agreement task and give an algorithm for solving it on the standard chromatic subdivision. In Section 5.4, if we iterate the standard chromatic subdivision a sufficient number of times, then there is a color and carrier-preserving map from the iterated standard chromatic subdivision to any chromatic subdivision. This claim implies that any simplex agreement protocol for the iterated chromatic subdivision yields a simplex agreement protocol for any chromatic subdivision.

5.2. THE CHROMATIC SUBDIVISION. We start with a purely combinatorial definition of the standard chromatic subdivision. This definition is analogous to the combinatorial definition of the standard barycentric subdivision in Definition 2.27. Let $S^n = (\vec{s}_0, \ldots, \vec{s}_n)$, where $id(\vec{s}_i) = P_i$.

Definition 5.3. In the *standard chromatic subdivision* of S^n, denoted $\chi(S^n)$, each n-simplex has the form $(\langle P_0, S_0\rangle, \ldots, \langle P_n, S_n\rangle)$, where S_i is a face of S^n, such that (1) $P_i \in ids(S_i)$, (2) for all S_i and S_j, one is a face of the other, and (3) if $P_j \in ids(S_i)$, then $S_j \subseteq S_i$.

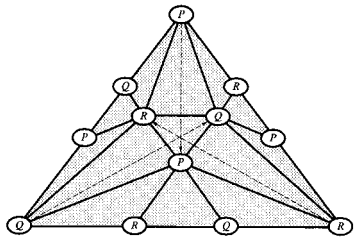

FIG. 33. Standard chromatic subdivision.

We refer to the $\vec{x}_i = \langle P_i, S^n \rangle$ as the *central vertexes* of the subdivision. The standard chromatic subdivision of S^2 is illustrated in Figure 33.

Definition 5.4. The *iterated* standard chromatic subdivision $\chi^K(S^n)$ is the result of iterating the standard chromatic subdivision K times.

It is sometimes useful to restrict subdivisions to a subcomplex.

Definition 5.5. Let $\mathscr{C}$ be a complex with subcomplexes $\mathscr{A}$ and $\mathscr{B}$ such that every simplex C of $\mathscr{C}$ can be written uniquely as $A \cdot B$ (the *join* of A and B), where A is a simplex of $\mathscr{A}$, and B of $\mathscr{B}$. (Either A or B could be empty.) Any subdivision γ of $\mathscr{A}$ induces a subdivision $\gamma(\mathscr{C}/\mathscr{B})$ of $\mathscr{C}$, called "γ of $\mathscr{C}$ holding $\mathscr{B}$ *fixed*", defined to be the complex of all joins $A' \cdot B$ where $A' \in \gamma(\mathscr{A})$, $B \in \mathscr{B}$, and carrier $(A', \mathscr{A}) \cdot B$ a simplex of $\mathscr{C}$.

For the standard chromatic subdivision, the process can be repeated: $\chi^2(\mathscr{C}/\mathscr{B}) = \chi(\chi(\mathscr{C}/\mathscr{B})/\mathscr{B})$, and so on.

LEMMA 5.6. *There is a color and carrier-preserving simplicial map*

$$\phi\colon \chi^K(\mathscr{C}) \to \chi^K(\mathscr{C}/\mathscr{B}).$$

PROOF. The map ϕ sends each vertex $\vec{v}$ of $\chi^K(\mathscr{C})$ to the unique vertex $\vec{u}$ in $\chi^K(\mathscr{A}/\mathscr{B})$ such that $id(\vec{v}) = id(\vec{u})$. □

We now give an equivalent geometric definition of the standard chromatic subdivision. A proof that the two definitions are equivalent is given by Hoest [1997]. To avoid notational clutter, we use $f\langle P, v\rangle$ as shorthand for $f(\langle P, v\rangle)$.

Definition 5.7. We construct the following homeomorphism $\iota\colon |\chi(S)| \to |S|$ inductively by dimension. Assume inductively that there exist homeomorphisms

$$\iota_i\colon |\chi(face_i(S))| \to |face_i(S)|.$$

Let $\vec{b} = \Sigma_{i=0}^{m}(\vec{s}_i/(m+1))$ be the barycenter of S, where $m = \dim(S)$, and δ any value such that $0 < \delta < 1/(m+1)$. Define for $id\ P_i$ and simplex R,

$$\iota\langle P_i, R\rangle = \begin{cases} \iota_i\langle P_i, R\rangle & \text{if} \quad R \subseteq face_i(S). \\ (1+\delta)\vec{b} - \delta\vec{s}_i & \text{if} \quad R = S. \end{cases}$$

See Figure 33.

For any value of δ such that $0 < \delta < 1/(m + 1)$, this definition gives an exact geometric construction for the chromatic subdivision. We will use this construction for the remainder of this section.

Definition 5.8. The *mesh* of a complex is the maximum diameter of any simplex.

LEMMA 5.9. *$mesh(\chi(S^n)) \leq n/(n + 1)\ diam(S^n)$.*

PROOF. We argue by induction on n. When n is zero, the claim is trivial. Let $\mathcal{S}^{n-1}$ be the boundary complex of S^n, $\beta(S^n)$ the barycentric subdivision, and $\vec{b}$ and $\vec{b}_i$ the respective barycenters of S^n and $face_i(S^n)$. Assume inductively that the claim holds for simplexes in $\chi(\mathcal{S}^{n-1})$. From Definition 5.7, each remaining central vertex $\vec{x}_i$ has the form $\vec{x}_i = (1 + \delta)\vec{b} - \delta\vec{s}_i$, which lies on the line joining $\vec{b}$ to $\vec{b}_i$. If $\vec{x} \in \chi(face_i(S^n))$, then the edge $(\vec{x}, \vec{x}_i)$ lies inside the triangle $(\vec{x}, \vec{b}, \vec{b}_i)$, which lies inside a simplex in $\beta(S^n)$. Since $mesh(\beta(S^n)) \leq (n/(n + 1))diam(S^n)$ [Munkres 1984, Theorem 15.4], $|\vec{x} - \vec{x}_i| \leq (n/(n + 1))diam(S^n)$. Finally, for any central vertex $\vec{x}_j$, $|\vec{x}_i - \vec{x}_j| = \delta|\vec{s}_i - \vec{s}_j|$, and the claim follows because $\delta < 1/(n + 1)$. □

Lemma 5.9 implies that by taking sufficiently large K, $mesh(\chi^K(\mathcal{I}))$ can be made arbitrarily small.

5.3. SIMPLEX AGREEMENT. Consider a task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ together with a subdivision σ and map $\mu: \sigma(\mathcal{I}) \to \mathcal{O}$ satisfying the conditions of the theorem. As described above, we will reduce any protocol to a "simplex agreement" protocol in which processes converge to the vertexes of a single simplex in $\sigma(\mathcal{I})$. More precisely:

Definition 5.10. Let $\mathcal{I}$ be an $(n + 1)$-colored complex, and σ a chromatic subdivision of $\mathcal{I}$. The *simplex agreement task* $\langle \mathcal{I}, \sigma(\mathcal{I}), \sigma \rangle$ has input complex $\mathcal{I}$, output complex $\sigma(\mathcal{I})$, and a task specification

$$\sigma = \{(S^m, T^m) | T^m \in \sigma(S^m)\}.$$

For brevity, "simplex agreement on $\sigma(\mathcal{I})$" means $\langle \mathcal{I}, \sigma(\mathcal{I}), \sigma \rangle$.

Given a task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ satisfying the conditions of Theorem 3.1, let $\sigma(\mathcal{I})$ and $\mu: \sigma(\mathcal{I}) \to \mathcal{O}$ be the subdivision and simplicial map guaranteed by the theorem. Any protocol that solves the simplex agreement task $\langle \mathcal{I}, \sigma(\mathcal{I}), \sigma \rangle$ can be adapted to solve $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ simply by applying μ to the result of the simplex agreement protocol.

Combining these observations yields a protocol for any task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ that satisfies the conditions of Theorem 3.1: Each process executes the simplex agreement protocol for $\chi^K(\mathcal{I})$, and applies ϕ and then μ to the result.

LEMMA 5.11. *There exists a wait-free protocol for simplex agreement on $\chi(\mathcal{I})$.*

PROOF. Each process P_i must choose a face of S_i of S^n such that (1) $P_i \in ids(S_i)$, (2) for all S_i and S_j, one is a subset of the other, and (3) if $P_j \in ids(S_i)$, then $S_j \subseteq S_i$. This is exactly the *participating set* problem of Borowsky and Gafni [1993], developed as part of their "immediate snapshot" algorithm. Their elegant wait-free solution appears in Figure 34. Borowsky [1995] gives a proof of this algorithm. □

```
procedure ParticipatingSet(int me, array F)
  array view_F[0..n] = {null, ..., null}
  repeat
    F[me] := F[me]-1;
    for i in 0 .. n do
      view_F[i] := F[i]
    S = {i | view_F[i] <= F[i]}
    until |S| >= F[me]
  return S;
```

FIG. 34. The Participating Set Protocol.

LEMMA 5.12. *There is a wait-free solution for simplex agreement on* $\chi^K(\mathcal{I})$, *for any* $K > 0$.

PROOF. Iterate the participating set algorithm [Borowsky and Gafni 1993] K times (Figure 35). □

5.4. MAPPING SUBDIVISIONS. In this section, we prove a number of lemmas about subdivisions. Recall from Definition 2.21 that any point $\vec{s}$ in $|\mathcal{I}|$ has a *barycentric representation*

$$\vec{s} = \sum_{i=0}^{n} s_i \cdot \vec{s}_i,$$

where each $0 \leq s_i \leq 1$, $\Sigma_i s_i = 1$, and the $\vec{s}_i$ span a simplex $\mathcal{I}$. The s_i are called the *barycentric coordinates* of $\vec{s}$ with respect to $\mathcal{I}$, and *carrier*$(\vec{s}, \mathcal{I})$ is the simplex of $\mathcal{I}$ spanned by the $\vec{s}_i$ for which $s_i > 0$.

Recall that a sequence of vertexes $\vec{s}_0, \ldots, \vec{s}_k$ is *affinely independent* if $\vec{s}_1 - \vec{s}_0, \ldots, \vec{s}_k - \vec{s}_0$ are linearly independent. (The vertexes of a simplex are affinely independent by definition.) Any sequence of vertexes $\vec{s}_0, \vec{s}_1, \ldots, \vec{s}_k$ determines a *hyperplane*, denoted *hyper*$(\vec{s}_0, \vec{s}_1, \ldots, \vec{s}_k)$, defined to be the set of points expressible as

$$\vec{s} = \sum_{i=0}^{k} s_i \cdot \vec{s}_i,$$

where $\Sigma_{i=0}^{k} s_i = 1$. A hyperplane's *dimension* is one less than the size of the smallest set of vertexes that determines that hyperplane. If $\vec{s}$ is a vertex of any chromatic subdivision of S^n, and $\vec{s}_i$ the vertex of S^n such that $id(\vec{s}) = id(\vec{s}_i)$, then $\vec{s}_i \in$ *carrier*$(\vec{s}, S^n)$. This terminology extends to simplexes in a natural way. Simplexes $A = (\vec{a}_0, \ldots, \vec{a}_k)$ and $B = (\vec{b}_0, \ldots, \vec{b}_\ell)$ are *affinely independent* if the sequence $\vec{a}_0, \ldots, \vec{a}_k, \vec{b}_0, \ldots, \vec{b}_\ell$ is affinely independent. Define the hyperplane *hyper*(A) to be *hyper*$(\vec{a}_0, \ldots, \vec{a}_k)$. If H is a hyperplane, and $\vec{a}$ a point of H, the *open* ϵ*-ball* around $\vec{a}$ in H is the set of points $\vec{b}$ in H such that $|\vec{a} - \vec{b}| < \epsilon$.

LEMMA 5.13. *Let H and K be hyperplanes, and* $\vec{a}$ *a point in H but not in K. For some* $\epsilon > 0$, *the open* ϵ*-ball around* $\vec{a}$ *does not intersect K.*

```
shared
  array F[1..k][0..n] = {n+2, ..., n+2};       // all n+2
  array vertex[0..k][0..n] = {null, ..., null} // all null

SimplexAgree(int me, vertex input, int k)
  array S[1..k] = {{}, ..., {}};
  vertex[0][me] = input;
  for i in 1 .. k do
    S[i] = ParticipatingSet(me, F[i]);
    vertex[i][me] = <i, {vertex[i-1][j] | j in S[i]}>
  return vertex[k][me];
```

FIG. 35. The Iterated Participating Set Protocol.

PROOF. $H \cap K$ is a closed subset of H, and its complement $H - K$ is open and nonempty, so there exists $\epsilon > 0$ such that $H - K$ contains the open ϵ-ball around $\vec{a}$ in H. □

Informally, an *ϵ-perturbation* of a subdivision is a new subdivision constructed by slightly displacing some set of vertexes within their respective carriers (Figure 32). Formally,

Definition 5.14. Let α be a subdivision of $\mathcal{S}$, and $\epsilon > 0$. A subdivision α_* of $\mathcal{S}$ is an *ϵ-perturbation* of α if there is an isomorphism ι: $\alpha(\mathcal{S}) \to \alpha_*(\mathcal{S})$ such that for every vertex $\vec{a}$ of $\alpha(\mathcal{S})$, $carrier(\vec{a}, \mathcal{S}) = carrier(\iota(\vec{a}), \mathcal{S})$ and $|\vec{a} - \iota(\vec{a})| < \epsilon$.

For brevity, when we speak of *perturbing* a vertex $\vec{a}$ by ϵ in a subdivision $\alpha(\mathcal{S})$, we mean constructing a new subdivision $\alpha_*(\mathcal{S})$ by replacing $\vec{a}$ with a $\vec{b}$ that lies in the open ϵ-ball around $\vec{a}$ in $carrier(\vec{a}, \mathcal{S})$.

We now show that any vertex of a subdivision can be perturbed by a sufficiently small amount. We exploit the following lemma from Munkres [1984, Lemma 15.2].

LEMMA 5.15. *If $\{K_i\}$ is a collection of complexes in Euclidean space, and if every $|K_i| \cap |K_j|$ is the polyhedron of a subcomplex of both K_i and K_j, then $\cup K_i$ is a complex.*

LEMMA 5.16. *Let α be a subdivision of $\mathcal{S}$, and $\vec{a}$ a vertex of $\alpha(\mathcal{S})$. There exists $\epsilon_0 > 0$ such that any perturbation of $\vec{a}$ by ϵ_0 in α yields an ϵ_0-perturbation α_* of $\alpha(\mathcal{S})$.*

PROOF. Let $C = carrier$ ($\vec{a}$, $\mathcal{S}$). Because $\alpha(C)$ is a manifold with boundary, by Lemma 2.36, $lk(\vec{a}, \alpha(C))$ is a sphere, and $\vec{a}$ is an interior point of $st^\circ(\vec{a}, \alpha(C))$. There thus exists $\epsilon > 0$ such that the open ϵ-ball around $\vec{a}$ in $|C|$ lies in $st^\circ(\vec{a}, \alpha(C))$. Let $A_0^{n-1}, \ldots, A_N^{n-1}$ be the $(n-1)$-simplexes of $lk(\vec{a}, \alpha(\mathcal{S}))$. For $0 \le i \le N$, each $\vec{a} \cdot A_i^{n-1}$ is a simplex of $\alpha(\mathcal{S})$, hence, $\vec{a}$ is affinely independent of A_i^{n-1}, and $\vec{a}$ does not lie on $hyper(A_i^{n-1})$. By Lemma 5.13, there exists $\varepsilon_i > 0$ such that every point $\vec{b}$ of $hyper(C)$ within ε_i of $\vec{a}$ does not lie on $hyper(A_i^{n-1})$. Let $\epsilon_0 = \min(\epsilon, \varepsilon_0, \ldots, \varepsilon_N)$. It follows that every $\vec{b}$ within ϵ_0 of $\vec{a}$ in $\alpha(C)$ lies within $st^\circ(\vec{a}, \alpha(C))$, and is affinely independent of $A_0^{n-1}, \ldots, A_N^{n-1}$. Each $\vec{b} \cdot A_i^{n-1}$ is thus a simplex, and $\vec{b} \cdot lk(\vec{a}, \alpha(\mathcal{S}))$ is a complex with polyhedron identical to the polyhedron of $\vec{a} \cdot lk(\vec{a}, \alpha(\mathcal{S})) = st(\vec{a}, \alpha(\mathcal{S}))$.

Let $\mathcal{A}$ be the complex consisting of all simplexes in $\alpha(\mathcal{I})$ that do not contain $\vec{a}$, and let $\mathcal{B} = \vec{b} \cdot lk(\vec{a}, \alpha(\mathcal{I}))$. The intersection $|\mathcal{A}| \cap |\mathcal{B}|$ is the polyhedron of the complex $lk(\vec{a}, \alpha(\mathcal{I}))$, so by Lemma 5.15, $\mathcal{A} \cup \mathcal{B}$ is a complex. Since $|\mathcal{A} \cup \mathcal{B}| = |\mathcal{I}|$, this complex is the desired subdivision $\alpha_*(\mathcal{I})$. □

LEMMA 5.17. *If α is a subdivision of $\mathcal{I}$, $\vec{x}$ a point of $|\alpha(\mathcal{I})|$, $A = carrier(\vec{x}, \alpha(\mathcal{I}))$, and $\vec{a}$ a vertex of A, then $\vec{x} \in st^\circ(\vec{a}, \alpha(\mathcal{I}))$.*

PROOF. If not, then $\vec{x}$ lies on a proper face of A that does not contain $\vec{a}$, contradicting the hypothesis that $A = carrier(\vec{x}, \alpha(\mathcal{I}))$. □

In the remainder of this section, let $\alpha(\mathcal{I})$ and $\beta(\mathcal{I})$ be two possible chromatic subdivisions of a complex $\mathcal{I}$, and $\mathcal{B}$ a subcomplex of $\beta(\mathcal{I})$. We focus on the relation between α and $\mathcal{B}$.

Definition 5.18. Subdivision α is a *chromatic cover* for $\mathcal{B}$ if every simplex B of $\mathcal{B}$ is covered by the open stars of vertexes of $\alpha(\mathcal{I})$ labeled with process *ids* from $ids(B)$:

$$(\forall B \in \mathcal{B})\, B \subset \bigcup_{id(\vec{a}) \in ids(B)} st^\circ(\vec{a}, \alpha(\mathcal{I})).$$

Definition 5.19. Simplexes A in $\alpha(\mathcal{I})$ and B in $\mathcal{B}$ are *mismatched* if $ids(A)$ and $ids(B)$ are disjoint, but $|A|$ intersects $|B|$.

LEMMA 5.20. *Subdivision α is a chromatic cover for $\mathcal{B}$ if and only if $\alpha(\mathcal{I})$ and $\mathcal{B}$ contain no mismatched simplexes.*

PROOF. Assume there are no mismatched simplexes: for all $A \in \alpha(\mathcal{I})$ and $B \in \mathcal{B}$, if $ids(A)$ and $ids(B)$ are disjoint, so are $|A|$ and $|B|$. Let $\vec{x}$ be a point of $|\mathcal{B}|$, $A = carrier(\vec{x}, \alpha(\mathcal{I}))$, and $B = carrier(\vec{x}, \mathcal{B})$. Since $|A|$ and $|B|$ intersect at $\vec{x}$, $ids(A)$ and $ids(B)$ must also intersect, and therefore A contains a vertex $\vec{a}$ such that $id(\vec{a}) \in ids(B)$. By Lemma 5.17, $\vec{x} \in st^\circ(\vec{a}, \alpha(\mathcal{I}))$. It follows that α is a chromatic cover for $\mathcal{B}$.

Assume that α is a chromatic cover for $\mathcal{B}$. If $|A|$ is a simplex of $\alpha(\mathcal{I})$, then for any vertex $\vec{a}$ of $\alpha(\mathcal{I})$ where $id(\vec{a}) \notin ids(A)$, $|A|$ does not intersect $st^\circ(\vec{a}, \alpha(\mathcal{I}))$. Because α is a chromatic cover for $\mathcal{B}$, if $ids(A)$ and $ids(B)$ are disjoint, then every point of $|B|$ lies in such an open star $st^\circ(\vec{a}, \alpha(\mathcal{I}))$, so $|A|$ and $|B|$ are disjoint. □

Next we show that we can perturb any vertex of any subdivision of $\mathcal{B}$ without introducing any additional mismatches.

LEMMA 5.21. *Let γ be a subdivision of $\mathcal{B}$, and $\vec{g}$ a vertex of $\gamma(\mathcal{B})$. There exists $\epsilon_1 > 0$ such that any perturbation of $\vec{g}$ by ϵ_1 yields an ϵ_1-perturbation γ_* of γ such that the number of mismatched simplexes between $\alpha(\mathcal{I})$ and $\gamma_*(\mathcal{B})$ is no greater than between $\alpha(\mathcal{I})$ and $\gamma(\mathcal{B})$.*

PROOF. Lemma 5.16 states that there exists $\epsilon_0 > 0$ such that γ remains a subdivision if we perturb $\vec{g}$ by ϵ_0. Let $\{G_i\}$ be the set of simplexes of $\gamma(\mathcal{B})$ that have $\vec{g}$ as a vertex, and let $\{A_{ij}\}$ be the following set of simplexes of $\alpha(\mathcal{I})$ that are not mismatched with G_i: $ids(G_i) \cap ids(A_{ij}) = \emptyset$ and $|G_i| \cap |A_{ij}| = \emptyset$. Let δ_{ij} be the minimum distance from any point of $|G_i|$ to any point of $|A_{ij}|$ (well

defined because both are closed sets). Define

$$\epsilon_1 = \min\left(\epsilon_0, \min_{ij} (\delta_{ij}) \right)$$

This minimum is well defined and positive because δ_{ij} ranges over a finite number of positive distances. Perturbing $\vec{g}$ by ϵ_1 yields a new subdivision γ_* with no additional mismatched simplexes. □

LEMMA 5.22. *If α is a chromatic cover for $\mathcal{B}$, then there is an ϵ-perturbation of $\chi\beta(\mathcal{I})$, carrying $\chi(\mathcal{B})$ to a subcomplex $\chi_*(\mathcal{B})$, such that α is a chromatic cover for $\chi_*(\mathcal{B})$. (Note that $\chi_*(\mathcal{B})$ is not necessarily a subdivision of $\mathcal{B}$, although it is isomorphic to $\chi(\mathcal{B})$.)*

PROOF. If $\alpha(\mathcal{I})$ is not a chromatic cover for $\chi(\mathcal{B})$, then by Lemma 5.20 there exist mismatched simplexes $C = (\vec{c}_0, \ldots, \vec{c}_c)$ of $\chi(\mathcal{B})$, and $A = (\vec{a}_0, \ldots, \vec{a}_a)$ of $\alpha(\mathcal{I})$. Pick C and A to have minimal dimensions a and c in the sense that no proper face of C intersects A, and vice-versa, and let $B = \mathit{carrier}(C, \mathcal{I})$ of dimension b. Because B is covered by open stars of the form $st^\circ(\vec{a}, \alpha(\mathcal{I}))$, where $\mathit{id}(\vec{a}) \in \mathit{ids}(B)$, and because A has minimal dimension, $\mathit{ids}(A) \subseteq \mathit{ids}(B) - \mathit{ids}(C)$, or $a \leq b - c - 1$.

Because C is a simplex of $\chi(B)$, C includes a central vertex of B whose carrier in $\mathcal{B}$ is B. By reindexing, let this vertex be $\vec{c}_c$. By Lemma 5.21, there exists $\epsilon_1 > 0$ such that any perturbation of $\vec{c}_c$ by ϵ_1 introduces no additional mismatches. Let $\epsilon_2 = \min(\epsilon, \epsilon_1)$.

Let $H = \mathit{hyper}(\vec{a}_0, \ldots, \vec{a}_a, \vec{c}_0, \ldots, \vec{c}_{c-1})$. H has dimension at most $a + c$, which is strictly less than b, so B contains a point $\vec{b}$ not in H. Let

$$\varepsilon = |\vec{b} - \vec{c}_c| / \epsilon_1$$

$$\vec{c} = (1 - \varepsilon) \cdot \vec{c}_c + \varepsilon \cdot \vec{b}$$

Because $\vec{c} \in \mathit{carrier}(\vec{c}_c, \mathcal{I}) = B$, replacing $\vec{c}_c$ by $\vec{c}$ in $\chi(\mathcal{B})$ yields an ϵ_1-perturbation of β with no additional mismatches. Let $C' = (\vec{c}_0, \ldots, \vec{c}_{c-1}, \vec{c})$. We claim that C' does not intersect A. Consider the barycentric coordinates of points of $|C'|$ with respect to C. Because C has minimal dimension, $(\vec{c}_0, \ldots, \vec{c}_{c-1})$ does not intersect A, so any point of $|C'|$ whose c-th barycentric coordinate is zero does not lie in $|A|$. By construction, $\vec{c}$ does not lie in the hyperplane H (or in $|A|$) and neither does any point whose c-th barycentric coordinate is positive.

We have constructed an ϵ-perturbation of $\mathcal{B}$, carrying $\chi(\mathcal{B})$ to $\chi_*(\mathcal{B})$, with strictly fewer mismatches with $\alpha(\mathcal{I})$. The claim now follows from a simple inductive argument. □

LEMMA 5.23. *If α is a chromatic cover for $\mathcal{B}$, then for all $\epsilon > 0$, and $K \geq 1$, $\alpha(\mathcal{I})$ is a chromatic cover for an ϵ-perturbation of $\chi^K(\mathcal{B})$.*

PROOF. Let $\epsilon_i = (1 - (1/2^i))\epsilon$. We show inductively that for any $1 \leq i \leq K$, α is a chromatic cover for an ϵ_i-perturbation of $\chi^i(\mathcal{B})$. For the base case, $\alpha(\mathcal{I})$ is a chromatic cover for $\mathcal{B}$ by construction. As induction hypothesis, assume $\alpha(\mathcal{I})$ is a chromatic cover for $\chi^i_*(\mathcal{B})$, an ϵ_i-perturbation of $\chi^i(\mathcal{B})$. Lemma 5.22

states that $\alpha(\mathcal{I})$ is a chromatic cover for an $\epsilon/(2^{i+1})$-perturbation $\chi_*^{i+1}(\mathcal{B})$ of $\chi(\chi_*^i(\mathcal{B}))$. Every vertex of $\chi_*^i(\mathcal{B})$ is displaced by at most ϵ_i from its corresponding vertex in $\chi^i(\mathcal{B})$, and the last perturbation adds at most $\epsilon/2^{i+1}$, yielding a final maximal displacement of $\epsilon_i + \epsilon/2^{i+1} = \epsilon_{i+1}$. which is an $((i + 1)\epsilon/K)$-perturbation of $\chi^{i+1}(\mathcal{B})$. We have shown that α is a chromatic cover for an ϵ_i-perturbation of χ_i, and the lemma follows because $\epsilon_i < \epsilon$, for all $i \geq 1$. □

Definition 5.24. Define the *extended star* $st^*(S, \mathcal{I})$ of a simplex S in $\mathcal{I}$ to be

$$st^*(S, \mathcal{I}) = \bigcup_{\vec{s} \in S} st(\vec{s}, \mathcal{I}).$$

The next two lemmas are left as an exercise for the reader.

LEMMA 5.25. *If α is a subdivision of $\mathcal{I}$, and A a simplex of $\alpha(\mathcal{I})$,*

$$diam(st^*(A, \alpha(\mathcal{I}))) \leq 3 \cdot mesh(\alpha(\mathcal{I})).$$

LEMMA 5.26. *If β is a subdivision of $\mathcal{B}$, and β_* an ϵ-perturbation of β, then $mesh(\beta_*(\mathcal{B})) < mesh(\beta(\mathcal{B})) + 2\epsilon$.*

We will need the following lemma from Spanier.

LEMMA 5.27 [SPANIER 1966, 2.1.25]. *A set of vertexes $\vec{v}_0, \ldots, \vec{v}_m$ belong to a common m-simplex of $\mathcal{I}$ if and only if*

$$\bigcap_{i=0}^{m} st^\circ(\vec{v}_i, \mathcal{I}) \neq \emptyset.$$

The next lemma is technical, but it encompasses most of the work needed to prove our main theorem.

LEMMA 5.28. *If α is a chromatic cover for $\mathcal{B}$, then there exist $K \geq 0$, a subdivision γ of $\mathcal{B}$, and color and carrier-preserving simplicial maps ξ and ψ, where*

$$\chi^K(\mathcal{B}) \xrightarrow{\xi} \gamma(\mathcal{B}) \xrightarrow{\psi} \alpha(\mathcal{I}) \tag{4}$$

such that every simplex X in $\gamma(\mathcal{B})$ includes a vertex $\vec{y}$ where

$$\vec{y} \in \bigcap_{\vec{x} \in X} st^\circ(\psi(\vec{x}), \alpha(\mathcal{I})). \tag{5}$$

PROOF. We argue by induction on the dimension of $\mathcal{B}$. In the base case, this dimension is zero. Because α is a chromatic cover for $\mathcal{B}$, each vertex $\vec{b}$ of $\mathcal{B}$ lies in $st^\circ(\vec{a}, \alpha(\mathcal{I}))$, for a unique $\vec{a}$ where $id(\vec{b}) = id(\vec{a})$. Define $K = 0$, ξ the identity map, γ the trivial subdivision that introduces no new vertexes, and $\psi(\vec{b}) = \vec{a}$.

For the induction hypothesis, assume the claim for subcomplexes of dimension less than i. Consider $\mathcal{B}$ of dimension i. Because α is a chromatic cover for $\mathcal{B}$, each i-simplex B of $\mathcal{B}$ has a covering by open sets $st^\circ(\vec{a}, \alpha(\mathcal{I}))$, where $id(\vec{a}) \in ids(B)$.

Let λ be the minimum Lebesgue number of any such covering (well defined because $\mathcal{B}$ is finite). By Lemma 5.9, we can pick K_0 large enough that

$mesh(\chi^{K_0}(\mathcal{B})) < \lambda/9$. By Lemma 5.23, $\chi^{K_0}(\mathcal{B})$ has a $(\lambda/9)$-perturbation $\chi_*^{K_0}(\mathcal{B})$ for which α is a chromatic cover. By Lemma 5.26, this perturbation adds at most $2\lambda/9$ to the diameter of any simplex in the subdivision, so $mesh(\chi_*^{K_0}(\mathcal{B})) < \lambda/3$. By Lemma 5.25, for every $X^i \in \chi_*^{K_0}(\mathcal{B})$,

$$diam(st^*(X^i, \chi_*^{K_0}(\mathcal{B}))) < 3 \cdot mesh(\chi_*^{K_0}(\mathcal{B})) < \lambda,$$

so $st^*(X^i, \chi_*^{K_0}(\mathcal{B})) \subset st^\circ(\vec{a}, \alpha(\mathcal{J}))$ for some vertex $\vec{a}$ in $\alpha(\mathcal{J})$ where $id(\vec{a}) \in ids(X^i)$. X^i has at least one vertex $\vec{x}$ such that

$$st(\vec{x}, \chi_*^{K_0}(\mathcal{B})) \subset st^\circ(\vec{a}, \alpha(\mathcal{J})) \tag{6}$$

for some $\vec{a}$ in $\alpha(\mathcal{J})$ where $id(\vec{a}) \in ids(X^i)$. Define $\psi(\vec{x}) = \vec{a}$. Let $\mathcal{B}_0$ be the subcomplex of $\chi_*^{K_0}(\mathcal{B})$ spanned by vertexes that satisfy Eq. (6), and $\mathcal{B}_1$ the subcomplex spanned by vertexes that do not. $\mathcal{B}_1$ has dimension at most $i - 1$ because each i-simplex of $\chi_*^{K_0}(\mathcal{B})$ includes at least one vertex in $\mathcal{B}_0$.

By the induction hypothesis, there exist $K_1 \geq 0$, a subdivision γ' of $\mathcal{B}_1$, and color and carrier-preserving simplicial maps ξ' and ψ

$$\chi^{K_1}(\mathcal{B}_1) \xrightarrow{\xi'} \gamma'(\mathcal{B}_1) \xrightarrow{\psi} \alpha(\mathcal{J}), \tag{7}$$

such that every simplex X_1 in $\gamma'(\mathcal{B}_1)$ includes a vertex $\vec{y}$ that

$$\vec{y} \in \bigcap_{\vec{x} \in X_1} st^\circ(\psi(\vec{x}), \alpha(\mathcal{J})). \tag{8}$$

Let $\gamma(\mathcal{B}) = \gamma'(\mathcal{B}_1/\mathcal{B}_0)$, the subdivision of $\mathcal{B}$ constructed by subdividing $\mathcal{B}_1$ by γ' while leaving $\mathcal{B}_0$ fixed. We have defined a color and carrier-preserving vertex map ψ: $\gamma(\mathcal{B}) \to \alpha(\mathcal{J})$. We now prove that ψ is simplicial. Every simplex X in $\gamma(\mathcal{B})$ is a join $X_0 \cdot X_1$, where X_0 is a simplex in $\mathcal{B}_0$ and X_1 in $\gamma'(\mathcal{B}_1)$. For every vertex $\vec{x}_0$ of X_0,

$$\vec{y} \in X_1 \subset st(\vec{x}_0, \chi_*^{K_0}(\mathcal{B})).$$

By Eq. (6),

$$st(\vec{x}_0, \chi_*^{K_0}(\mathcal{B}) \subset st^\circ(\psi(\vec{x}_0), \alpha(\mathcal{J})).$$

Combining these equations with Eq. (8),

$$\vec{y} \in \bigcap_{\vec{x} \in X} st^\circ(\psi(\vec{x}), \alpha(\mathcal{J})).$$

By Lemma 5.27, the $\psi(\vec{x})$ span a vertex of $\alpha(\mathcal{J})$, so ψ is a simplicial map. So far we have shown that there exists a simplicial map

$$\gamma(\mathcal{B}) \xrightarrow{\psi} \alpha(\mathcal{J})$$

such that every simplex X in $\gamma(\mathcal{B})$ includes a vertex $\vec{y}$ that

$$\vec{y} \in \bigcap_{\vec{x} \in X} st^{\circ}(\psi(\vec{x}), \alpha(\mathcal{I})).$$

To complete the proof, we show that

$$\xi' \colon \chi^{K_1}(\mathcal{B}_1) \rightarrow \gamma'(\mathcal{B}_1)$$

can be extended to

$$\xi'' \colon \chi^{K_1}(\mathcal{B}/\mathcal{B}_0) \rightarrow \gamma(\mathcal{B})$$

by making ξ'' the identity on vertexes of $\mathcal{B}_0$. If X_0 is a simplex of $\mathcal{B}_0$, and X_1 in $\chi^{K_1}(\mathcal{B}_1)$, then the join $X_0 \cdot X_1$ is in $\chi^{K_1}(\mathcal{B}/\mathcal{B}_0)$ if and only if *carrier*$(X_1, \chi_*^{K_0}(\mathcal{B})) \cdot X_0$ is a simplex of $\chi_*^{K_0}(\mathcal{B})$. Similarly, if Y_1 is a simplex of $\gamma'(\mathcal{B}_1)$, $X_0 \cdot Y_1$ is in $\gamma(\mathcal{B}) = \gamma'(\mathcal{B}/\mathcal{B}_0)$ if and only if *carrier*$(Y_1, \chi_*^{K_0}(\mathcal{B})) \cdot X_0$ is a simplex of $\chi_*^{K_0}(\mathcal{B})$. By the induction hypothesis, ξ' is carrier-preserving, so *carrier*$(X_1, \chi_*^{K_0}(\mathcal{B})) =$ *carrier*$(\xi'(X_1), \chi_*^{K_0}(\mathcal{B}))$, and therefore ξ'' is a simplicial map.

By Lemma 5.6, there is a color and carrier-preserving simplicial map

$$\chi^{K_1}(\chi_*^{K_0}(\mathcal{B})) \rightarrow \chi^{K_1}(\mathcal{B}/\mathcal{B}_0)$$

and an isomorphism

$$\chi^{K_0+K_1}(\mathcal{B}) \rightarrow \chi^{K_1}(\chi_*^{K_0}(\mathcal{B})).$$

Composing these maps yields a color and carrier-preserving simplicial map

$$\xi \colon \chi^{K_0+K_1}(\mathcal{B}) \rightarrow \gamma(\mathcal{B}).$$

completing the proof of Eq. 4. □

THEOREM 5.29. *If σ is a chromatic subdivision of a complex $\mathcal{I}$, then there exists $K \geq 0$ and a color and carrier-preserving simplicial map*

$$\phi \colon \chi^{K}(\mathcal{I}) \rightarrow \sigma(\mathcal{I}).$$

PROOF. Note that $\sigma(\mathcal{I})$ is a chromatic cover for $\mathcal{I}$, so by Lemma 5.28, there exists $K > 0$ and a color and carrier-preserving simplicial maps

$$\chi^{K}(\mathcal{I}) \stackrel{\xi}{\rightarrow} \gamma(\mathcal{I}) \stackrel{\psi}{\rightarrow} \sigma(\mathcal{I})$$

Define $\phi \colon \chi^{K}(\mathcal{I}) \rightarrow \sigma(\mathcal{I})$ to be the composition of ξ and ψ. The map ϕ is simplicial, color-preserving, and carrier-preserving. □

THEOREM 5.30. *A decision task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ has a wait-free protocol using read-write memory if there exists a chromatic subdivision $\sigma(\mathcal{I})$ and a color-preserving simplicial map*

$$\mu \colon \sigma(\mathcal{I}) \rightarrow \mathcal{O}$$

such that for each simplex S in $\sigma(\mathcal{I})$, $\mu(S) \in \Delta($carrier$(S, \mathcal{I}))$.

PROOF. Suppose the participating processes start with inputs given by simplex S^m. By Theorem 5.29, there exists a color and carrier-preserving map

$$\phi\colon \chi^K(S^m) \rightarrow \sigma(S^m).$$

By hypothesis, there exists a simplicial map:

$$\mu\colon \sigma(S^m) \rightarrow \Delta(S^m).$$

The protocol has the following steps:

(1) Use the iterated participating set protocol to agree on a simplex in $\chi^K(S^m)$.
(2) A process that chooses vertex $\vec{x} \in \chi^K(S^m)$ then chooses as its output the value labeling $\mu(\phi(\vec{x}))$. □

This completes the proof of the Asynchronous Computability Theorem.

Our construction shows that we can assume without loss of generality that the span σ is an iterated chromatic subdivision.

COROLLARY 5.31. *A decision task* $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ *has a wait-free protocol using read-write memory if and only if there exists a* $K \geq 0$ *and a color-preserving simplicial map*

$$\mu\colon \chi^K(\mathcal{I}) \rightarrow \mathcal{O}$$

such that for each simplex S *in* $\chi^K(\mathcal{I})$, $\mu(S) \in \Delta(\mathit{carrier}(S, \mathcal{I}))$.

6. *Renaming with a Small Number of Names*

In the renaming task of Attiya et al. [1990] and Attiya and Welch [1998], processes are issued unique input names from a large name space, and must choose unique output names taken from a smaller name space. To rule out trivial solutions, protocols must be *anonymous* [Sect. 17.3], meaning that the value any process chooses does not depend in any way on the value of any participant's process id (including its own). Informally, a process may choose its output value based only on the input name it received, and on how its memory accesses are interleaved with the memory accesses of the other processes. (We give a formal definition of anonymity below.)

In the message-passing model, Attiya et al. [1990] showed that renaming has a wait-free solution when $K \geq 2n + 1$, and none when $K \leq n + 2$. Bar-Noy and Dolev [1989] extended their upper bound solution to the shared read-write memory model. Whether a protocol exists for $n + 2 < K \leq 2n$ names remained open until 1993, when Herlihy and Shavit [1993] showed that no such protocol exists. Henceforth, by *renaming*, we mean the renaming task where $K \leq 2n$.

The restriction to anonymous protocols implies that the asynchronous computability theorem does not apply. Nevertheless, a variant of this theorem can be devised for anonymous protocols. In this section we show how to adapt the asynchronous computability theorem to prove that the renaming task has no anonymous wait-free read/write protocol. This section differs from previous sections in the level of technical detail: we make explicit use of elementary homology theory (as presented by Munkres [1984]). Our original proof appeared in 1993 [Herlihy and Shavit 1993]. Here we present a simplified version of that

proof, based on the chain map proof methodology developed by Herlihy and Rajsbaum [1995].

6.1. PROOF OUTLINE. The key insight underlying our proof is that the anonymity requirement makes it impossible to break symmetry among certain input configurations. We capture and formalize this intuition through the following sequence of steps.

—We begin by giving a formal statement of the notion of anonymity [Attiya and Welch 1998]. We prove a variant of the asynchronous computability theorem for anonymous protocols. This theorem states that the map from a subdivision of the input complex to the output complex must satisfy certain symmetry properties. The theorem statement exploits the observation of Corollary 5.31 that we can restrict our attention to the iterated standard chromatic subdivision.

—We reduce the renaming task itself to a *reduced renaming* task in which processes choose Boolean values instead of names. The resulting output complex is smaller and easier to work with. In particular, the complex has a single "hole" which will act as an obstruction to any protocol. For a larger number of names, the corresponding output complex has no hole, so the existence of this hole characterizes the boundary between solvable and unsolvable instances of renaming.

—We identify a subcomplex of the input complex satisfying two properties: the subcomplex itself is "solid", with no holes, and the inputs along its boundary are symmetric.

—We then apply the anonymous computability theorem to show that the simplicial map guaranteed by the theorem wraps the symmetric boundary of the solid subcomplex around the output complex's hole a nonzero number of times, a contradiction.

6.2. ANONYMITY. We now give a formal definition of the notion of anonymity used by Attiya and Welch [1998] to define the renaming task. Let π be any permutation of $0, \ldots, n$. The permutation π acts on any labeled simplex by replacing each occurrence of a process id P in the label with the process id $\pi(P)$. Here are some examples.

—For an input or output simplex S^n, π sends each vertex $\langle P, v\rangle$ to $\langle \pi(P), v\rangle$.

—For the chromatic subdivision $\chi(S^n)$, π sends each vertex $\langle P, S\rangle$ to $\langle \pi(P), \pi(S)\rangle$, where S is a face of S^n. The reader is invited to check that $\pi(\chi(S)) = \chi(\pi(S))$.

—For a protocol complex $\mathcal{P}(\mathcal{I})$, π sends each vertex $\langle P, e\rangle$ to $\langle \pi(P), \pi(e)\rangle$, where $\pi(e)$ denotes the execution view in which each process id Q is replaced by $\pi(Q)$.

Definition 6.1. A complex $\mathcal{C}$ is *symmetric* if π induces a simplicial map from $\mathcal{C}$ to itself, denoted $\pi: \mathcal{C} \to \mathcal{C}$. If $\mathcal{A}$ and $\mathcal{B}$ are symmetric complexes, then $\phi: \mathcal{A} \to \mathcal{B}$ is *symmetric under permutation* if $\pi(\phi(\vec{v})) = \phi(\pi(\vec{v}))$ for any permutation π. A task specification $\langle \mathcal{I}, \mathcal{O}, \Delta\rangle$ is *symmetric* if $\mathcal{I}$ and $\mathcal{O}$ are symmetric, and for all $S^n \in \mathcal{I}$, $\Delta(\pi(S^n)) = \pi(\Delta(S^n))$.

In short, the problem specification depends only on input values, not process ids. This restriction is weak: all tasks considered in this paper (excluding our contrived quasi-consensus example of Section 3.2) have symmetric specifications.

In Section 5, we showed that any protocol can be expressed as an iterated "immediate snapshot" algorithm. This kind of protocol has the property that in any valid execution, replacing each process id P with $\pi(P)$ yields another valid execution. As a result, we can assume without loss of generality that any protocol complex is symmetric.

Definition 6.2. A protocol $\mathcal{P}$ is *anonymous* if the decision map δ is symmetric under permutation: for every simplex T in $\mathcal{P}(S^n)$, $\pi(\delta(T)) = \delta(\pi(T))$.

We now give the anonymous variant of the asynchronous computability theorem. The proof appears in the appendix, but it is essentially the same as the proof of the original, except that we use a specific symmetric subdivision based on the standard chromatic subdivision.

THEOREM 6.3 (ANONYMOUS COMPUTABILITY THEOREM). *A symmetric decision task* $\langle \mathcal{I}^n, \mathcal{O}^n, \Delta \rangle$ *has a wait-free anonymous protocol using read-write memory if and only if there exists an integer K and a color-preserving simplicial map*

$$\mu\colon \chi^K(\mathcal{I}) \rightarrow \mathcal{O}$$

symmetric under permutation, such that for each simplex X in $\chi^K(\mathcal{I})$, $\mu(X) \in \Delta(carrier(X, \mathcal{I}))$.

6.3. REDUCED RENAMING. We now simplify the task by reducing the size of the output complex. Consider the following *reduced renaming* task.

REDUCED RENAMING. Each process chooses a binary value, and in every execution where all $n + 1$ processes choose a value, at least one chooses 0, and at least one chooses 1.

LEMMA 6.4. *If an anonymous renaming protocol exists, then so does an anonymous reduced renaming protocol.*

PROOF. Any renaming protocol (anonymous or not) can be transformed into a reduced renaming protocol simply by taking the parity of the names chosen. If the original protocol was anonymous, so is the reduced protocol. □

For example, the annulus of Figure 36 is the reduced output for the original three-process torus-shaped output complex of Figure 8.

6.4. A THREE-PROCESS EXAMPLE. The full proof of the renaming lower bound is unavoidably technical, due to the need to reason formally about the structure of high-dimensional complexes. Nevertheless, the need for formal rigor should not be allowed to obscure the inherent geometric simplicity of the proof's essence. In this section, we describe a three-process example that illustrates the basic ideas underlying the full proof.

The output complex $\mathcal{O}^2$ for three-process reduced renaming, shown in Figure 36, is the annulus constructed by taking the binary 2-sphere over P, Q, and R, and removing the all-zero and all-one simplexes. As we will explain, the resulting hole

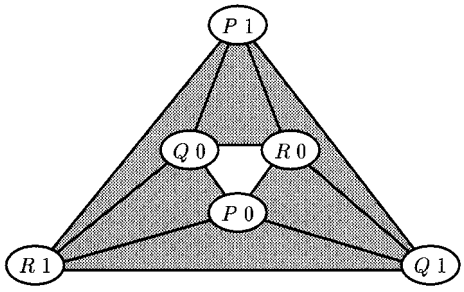

FIG. 36. Reduced renaming output complex $\mathbb{O}^2$.

will act as an "obstruction" to any anonymous protocol. The boundary between four and five names is exactly the boundary between a reduced renaming task whose output complex has a hole (impossible) and one that does not (possible). For notational convenience, we will use $P0$ as shorthand for $\langle P, 0\rangle$, and so on.

Assume we have an anonymous protocol for three-process reduced renaming. Consider the subcomplex $\mathcal{T}$ of the input complex shown in Figure 37. This subcomplex is isomorphic to $\chi(S^2)$, the standard chromatic subdivision of a 2-simplex. The anonymous computability theorem states that there is a simplicial map μ carrying a subdivision of $\mathcal{T}^2$ to the reduced renaming output complex $\mathbb{O}^2$. We will argue that any such map must wrap the boundary of $\mathcal{T}^2$ around the hole in $\mathbb{O}^2$ a nonzero number of times. As stated formally in Section 6.6, no continuous map can carry the boundary of a "solid" region ($\mathcal{T}^2$) to the boundary of a hole, so we have a contradiction.

The inputs along the boundary of $\mathcal{T}^2$ are symmetric in the following sense. Let $\mathcal{T}_P$, $\mathcal{T}_Q$, and $\mathcal{T}_R$ denote the subdivided edges opposite the "corner" vertexes $P0$, $Q0$, and $R0$. Informally, traversing any $\mathcal{T}_X$ from the smaller id toward the larger id, we encounter the same sequence of input values.[7] More precisely, let π_P and π_Q be the following permutations:

	P	Q	R
π_P	Q	R	P
π_Q	P	R	Q.

The symmetry property is that $\pi_P(\mathcal{T}_R) = \mathcal{T}_P$ and $\pi_Q(\mathcal{T}_R) = \mathcal{T}_Q$.

The anonymous computability theorem guarantees a simplicial map μ: $\chi^K(\tau(S^2)) \to \mathbb{O}^2$ symmetric under permutation. Assume, without loss of generality, that if P runs solo, it chooses an even name: $\mu(P0) = P0$. Because the protocol is anonymous, $\mu(Q0) = Q0$. The map μ carries the subdivided edge $\chi^K(\mathcal{T}_R)$ to a path linking $P0$ to $Q0$ in $\mathbb{O}^2$. For the sake of this example, let us assume that

$$\mu(\mathcal{T}_R) = (P0, Q1, P1, Q0), \tag{9}$$

as illustrated in Figure 38.

Because the protocol is anonymous, μ maps $\mathcal{T}_P$ and $\mathcal{T}_R$ to $\mathbb{O}$ in a symmetric

[7] Note that vertex labels in $\mathcal{T}^2$ are not necessarily unique.

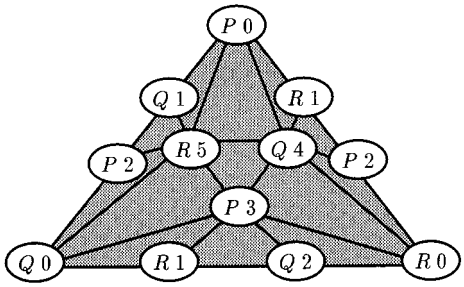

FIG. 37. Input subcomplex $\mathcal{T}^2$.

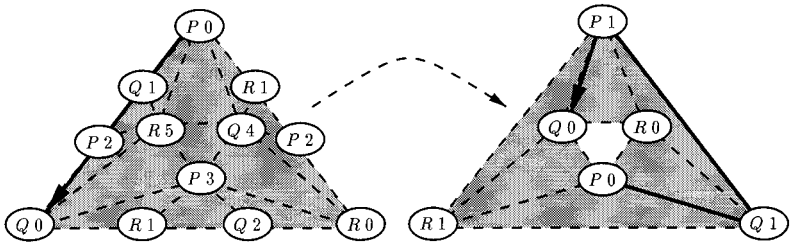

FIG. 38. Mapping an edge of the input complex.

way. More precisely,

$$\begin{aligned}\mu(\mathcal{T}_P) &= \mu(\pi_P(\mathcal{T}_R)) \\ &= \pi_P(\mu(\mathcal{T}_R)). \\ &= \pi_P(P0, Q1, P1, Q0) \\ &= (Q0, R1, Q1, R0). \end{aligned} \tag{10}$$

By similar reasoning,

$$\mu(\mathcal{T}_Q) = (R0, P1, R1, P0). \tag{11}$$

Combining Eqs. (9), (10), and (11) shows that μ wraps the boundary complex of $\mathcal{T}^2$ around the hole in $\mathbb{O}^2$ twice in the counter-clockwise direction, as illustrated in Figure 39.

No matter what path we choose for $\mu(\mathcal{T}_R)$, we will see that μ wraps the boundary of $\mathcal{T}^2$ around the hole $3k + 1$ times, for some integer value k. (A positive k corresponds to a clockwise orientation, and a negative value is counter-clockwise.) In this example, $k = -1$.

Although this example does not constitute a proof, all the key elements of the full proof are represented. The reduced renaming protocol for $n + 1$ processes and $2n$ names also has a hole, corresponding to the "missing" all-zero simplex 0^n. (As before, the boundary between $2n$ and $2n + 1$ names is the boundary between an output complex with a hole, and one without.) We construct an input subcomplex $\mathcal{T}^n$, isomorphic to $\chi(S^n)$, such that input names are symmetric on the boundary of $\mathcal{T}^n$. We formalize the claim that μ cannot wrap the boundary of $\mathcal{T}$ around the hole in $\mathbb{O}^n$ a nonzero number of times. We then exploit the

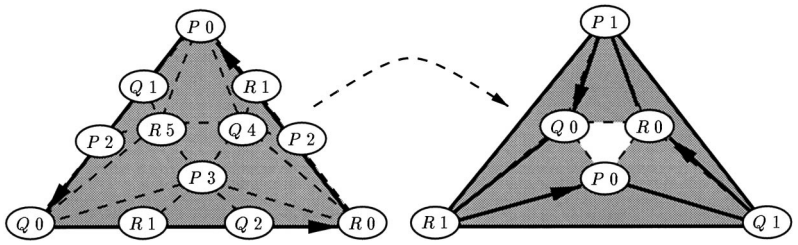

FIG. 39. Mapping the boundary of the input complex.

symmetry of $\mathcal{T}^n$ and μ to prove that μ does indeed map the boundary of $\mathcal{T}^n$ around the hole $(n + 1)k + 1$ times, for some integer value k (again, positive and negative values indicate distinct orientations). The value $(n + 1)k + 1$ is nonzero for any k, yielding the desired contradiction.

6.5. A SYMMETRIC INPUT SUBCOMPLEX. We now formalize the construction of the "symmetric input subcomplex" $\mathcal{T}^n$ illustrated (for $n = 2$) in Figure 37.

Definition 6.5. For $0 \le m \le n$, and $0 \le i \le m$, define the permutation π_i^m to be

$$\pi_i^m(k) = \begin{cases} k & \text{if} \quad 0 \le k < i \\ k + 1 & \text{if} \quad i \le k < m \\ i & \text{if} \quad k = m. \end{cases}$$

The permutations π_i^n are just the generalizations of π_P and π_Q defined over $\mathcal{T}^2$ in the previous section.

LEMMA 6.6. *For* $i < j$, $0 \le k \le m - 2$,

$$\pi_i^m(\pi_{j-1}^{m-1}(k)) = \pi_j^m(\pi_i^{m-1}(k)).$$

PROOF. Both composite permutations map $0, \ldots, m - 1$ to $0, \ldots, \hat{i}, \ldots, \hat{j}, \ldots, m$, respectively. □

The symmetric input subcomplex T^n is isomorphic to the standard chromatic subdivision of an n-simplex by a color-preserving isomorphism denoted by

$$\iota: \chi(S^n) \to \mathcal{T}^n.$$

Let $\mathcal{T}_i^{n-1}$ be the subcomplex $\iota(face_i(S^n))$, which we call the ith *face* of $\mathcal{T}^n$. We require that the faces of $\mathcal{T}^n$ satisfy the following symmetry property:

$$\pi_i^n(\mathcal{T}_n^{n-1}) = \mathcal{T}_i^{n-1}. \tag{12}$$

The subcomplex $\mathcal{T}^n$ is constructed inductively. Complex $\mathcal{T}^0$ is the single vertex $\langle P_0, 0\rangle$. Assume inductively that we have constructed T^{m-1}, for $0 < m \le n$, colored with process ids $P_0, \ldots, P_{m-1}$, satisfying Eq. 12, using $m(m + 1)/2$ distinct input values. We construct $\mathcal{T}^m$ by assigning input values to $\chi(S^m)$. As an operator, each permutation π_i^m defines an isomorphism

$$\pi_i^m: \mathcal{T}^{m-1} \to \mathcal{T}_i^{m-1}.$$

Assign each vertex in $\mathcal{T}_i^{m-1}$ the value of its matching vertex in $\mathcal{T}^{m-1}$, and assign each central vertex of $\mathcal{T}^m$ labeled with P_i the input value $m(m+1)/2 - m + i = m(m-1)/2 + i$.

To show that this assignment of values is well defined, we must check that the value assigned to a face vertex does not depend on the choice of π_i^m.

When $m = 0$, all face vertexes are assigned 0, Assume $m > 0$. For permutations π_i^m and π_j^m, where $i < j$,

$$\begin{aligned}\pi_i^m(T^{m-1}) \cap \pi_j^m(T^{m-1}) &= \pi_i^m(T_{j-1}^{m-1}) \\ &= \pi_i^m(\pi_{j-1}^{m-1}(T^{m-2})).\end{aligned}$$

By a similar argument,

$$\pi_i^m(T^{m-1}) \cap \pi_j^m(T^{m-1}) = \pi_j^m(\pi_i^{m-1}(T^{m-2})).$$

By Lemma 6.6, $\pi_i^m(\pi_{j-1}^{m-1}(T^{m-2})) = \pi_j^m(\pi_i^{m-1}(T^{m-2}))$, so the value assigned is independent of π_i.

6.6. Orientation, Cycles, and Boundaries. This section reviews some standard technical definitions needed for our proof. Our discussion closely follows that of Munkres [1984, Section 1.13], which the reader is encouraged to consult for more detail.

Let $S^n = (\vec{s}_0, \ldots, \vec{s}_n)$ be an n-simplex. An *orientation* for S^n is an equivalence class of orderings on $\vec{s}_0, \ldots, \vec{s}_n$, consisting of one particular ordering and all even permutations of it. For example, an orientation of a 1-simplex $(\vec{s}_0, \vec{s}_1)$ is just a direction, either from $\vec{s}_0$ to $\vec{s}_1$, or vice-versa. An orientation of a 2-simplex $(\vec{s}_0, \vec{s}_1, \vec{s}_2)$ can either be clockwise, as in $(\vec{s}_0, \vec{s}_1, \vec{s}_2)$, or counterclockwise $(\vec{s}_0, \vec{s}_2, \vec{s}_1)$. Any orientation of a simplex induces orientations on its faces. An n-manifold with boundary is *oriented* if each n-simplex is oriented in a way that each internal $(n-1)$-simplex inherits opposite orientations from its two containing n-simplexes. Any n-sphere can be oriented in this way. Given a simplex whose vertexes are colored by process ids, the *standard orientation* orders them by increasing process id.

A *d-chain* is a formal sum of oriented d-simplexes: $\Sigma_{i=0}^{\ell} \lambda_i \cdot S_i^d$, where λ_i is an integer. When writing chains, we typically omit d-simplexes with zero coefficients, unless they are all zero, when we simply write 0. We write $1 \cdot S^d$ as S^d, $-1 \cdot S^d$ as $-S^d$, and $S^d + \cdots + S^d$ (k times) as $k \cdot S^d$. It is convenient to identify $-S^d$ with S^d having the opposite orientation. The q-chains of $\mathcal{K}$ form a free Abelian group $C_q(\mathcal{K})$, called the qth *chain group* of $\mathcal{K}$.

Let $S^d = (\vec{s}_0, \ldots \vec{s}_d)$ be an oriented d-simplex. Define $face_i(S^d)$, the ith *face* of S^d, to be the $(d-1)$-simplex $(\vec{s}_0, \ldots, \hat{s}_i, \ldots, \vec{s}_d)$, where the circumflex denotes omission. The *boundary* operator ∂_q: $C_q(\mathcal{K}) \to C_{q-1}(\mathcal{K})$, $q > 0$, is defined on simplexes:

$$\partial_q S^q = \sum_{i=0}^{q} (-1)^i \cdot face_i(S^q),$$

and extends additively to chains: $\partial_q(\alpha_0 + \alpha_1) = \partial_q\alpha_0 + \partial_q\alpha_1$. The boundary operator has the important property that applying it twice causes chains to vanish:

$$\partial_{q-1}\partial_q\alpha = 0. \tag{13}$$

(Henceforth, we omit subscripts from boundary operators.) A q-chain α is a *boundary* if $\alpha = \partial\beta$ for some $(q + 1)$-chain β, and it is a *cycle* if $\partial\alpha = 0$. Equation 13 implies that that every boundary is a cycle.

Definition 6.7. Two d-cycles α and β are *homologous*, written $\alpha \sim \beta$, if $\alpha - \beta$ is a boundary.

The *chain complex* $C(\mathcal{K})$ is the sequence of groups and homomorphisms $\{C_q(\mathcal{K}), \partial\}$. Let $C(\mathcal{K}) = \{C_q(\mathcal{K}), \partial\}$ and $C(\mathcal{L}) = \{C_q(\mathcal{L}), \partial'\}$ be chain complexes for simplicial complexes $\mathcal{K}$ and $\mathcal{L}$. A *chain map* ϕ: $C(\mathcal{K}) \to C(\mathcal{L})$ is a family of homomorphisms

$$\phi_q\colon C_q(\mathcal{K}) \to C_q(\mathcal{L}),$$

that preserve cycles and boundaries: $\partial' \circ \phi_q = \phi_{q-1} \circ \partial$. Any subdivision σ of $\mathcal{K}$ induces a chain map in the obvious way,

$$C(\mathcal{K}) \to C(\sigma(\mathcal{K})),$$

as does any simplicial map $\phi\colon \mathcal{K} \to \mathcal{L}$

$$C(\mathcal{K}) \to C(\mathcal{L}),$$

or permutation operator $\pi\colon \mathcal{K} \to \mathcal{K}$

$$C(\mathcal{K}) \to C(\mathcal{K}).$$

We use the following facts [Munkres 1984, Th.8.3] about chain groups of spheres. Let the complex $\mathcal{B}^{n-1}$ be an $(n - 1)$-sphere, and let B be the cycle constructed by orienting the simplexes of $\mathcal{B}^{n-1}$.

LEMMA 6.8. *Every $(n - 1)$-cycle of $\mathcal{B}^{n-1}$ is homologous to $k \cdot B$, for some integer k, and every q-cycle is a boundary, for $q < n - 1$.*

We refer to B as a *generator* for $\mathcal{B}^{n-1}$.
Let $\mathcal{A} \subset \mathcal{K}$ be complexes.

Definition 6.9. A *deformation retraction* of $\mathcal{K}$ onto $\mathcal{A}$ is a continuous map F: $|\mathcal{K}| \times I \to |\mathcal{K}|$ such that $F(x, 0) = x$ for $x \in |\mathcal{K}|$, $F(x, 1) \in \mathcal{A}$ for $x \in |\mathcal{K}|$, and $F(a, t) = a$ for $a \in |\mathcal{A}|$. If such an F exists, we say that $\mathcal{A}$ is a *deformation retract* of $\mathcal{K}$. Every cycle of $\mathcal{K}$ is homologous to a cycle of $\mathcal{A}$.

LEMMA 6.10. *If the $(n - 1)$-sphere $\mathcal{B}$ is a deformation retract of $\mathcal{K}$, then every q-cycle of $\mathcal{K}$ is a boundary, and every $(n - 1)$-cycle of $\mathcal{K}$ is homologous to $k \cdot B$, for some integer k.*

We also refer to B as a *generator* of $\mathcal{K}$. If B and B' are both generators of $\mathcal{K}$, then $B \sim \pm B'$.

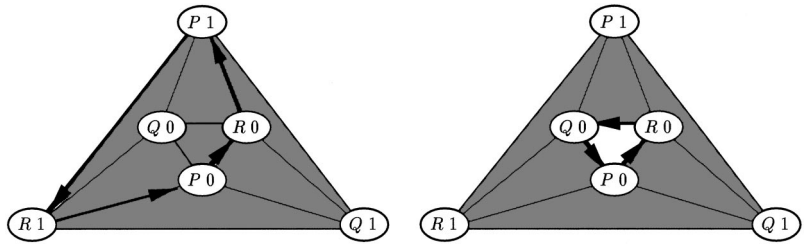

FIG. 40. Generators B_Q and $\partial 0^2$.

This paragraph is an aside for readers familiar with standard algebraic topology. The qth simplicial homology group $H_q(\mathcal{K})$ for a complex $\mathcal{K}$ is the quotient group of the q-dimensional group of cycles by the q-dimensional group of boundaries. Lemma 6.8 states the well-known fact that the homology groups of the $(n-1)$-sphere $\mathcal{B}^{n-1}$ are trivial below $(n-1)$, and infinite cyclic in dimension n, generated by B. Lemma 6.10 is a special case of the classical theorem [Munkres 1984, p. 108] that any complex has the same homology as its deformation retracts.

6.7. IMPOSSIBILITY OF RENAMING. Recall that reduced renaming requires each process to choose a binary value, but $n+1$ processes may not all choose 1, or all choose 0. The output complex $\mathbb{O}^n$ is thus constructed by taking a binary n-sphere and removing the all-zero simplex 0^n and the all-one simplex 1^n. Let $\dot{0}^{n-1}$ be the boundary complex of 0^n with the orientation induced by the standard orientation of 0^n. This complex is an $(n-1)$-sphere with generator ∂S^n. Because $\dot{0}^{n-1}$ is a deformation retract of $\mathbb{O}^n$, ∂S^n is also a generator for $\mathbb{O}^n$.

Let $\mathbb{O}_i^{n-1}$ be the subcomplex of $\mathbb{O}^n$ colored with process ids $P_0, \ldots, \hat{P}_i, \ldots, P_n$. This complex is a binary $(n-1)$-sphere. Let B_i denote the chain constructed by orienting each $(n-1)$-simplex of $\mathbb{O}_i^{n-1}$ so that the all-zero simplex has standard orientation. Because $\mathbb{O}_i^{n-1}$ is a deformation retract of $\mathbb{O}^n$, B_i is also a generator of $\mathbb{O}^n$. These generators are shown in Figure 40.

Algebraically, these generators are related as follows:

LEMMA 6.11. $B_i \sim (-1)^i \partial 0^n$.

PROOF. Since B_i and $\partial 0^n$ are both generators for $\mathbb{O}^n$, $B_i \sim \pm \partial 0^n$. The simplex $face_i(0^n)$ is common to both B_i and $\partial 0^n$. It has the standard orientation in B_i, and $(-1)^i$ times the standard orientation in $\partial 0^n$. □

Recall that subdivisions, permutation operators, and simplicial maps all induce chain maps. The standard chromatic subdivision χ induces a chain map

$$C(S^n) \rightarrow C(\chi(S^n)).$$

The isomorphism $\iota: \chi(S^n) \rightarrow \mathcal{T}^n$ (where $\mathcal{T}^n$ is the symmetric input subcomplex constructed in Section 6.5) induces a chain map

$$C(\chi(S^n)) \rightarrow C(\mathcal{T}^n).$$

Assume by way of contradiction that there exists an anonymous protocol for reduced renaming. The anonymous computability theorem (Theorem 6.3) gives a simplicial map $\mu: \chi^K(\mathcal{T}^n) \rightarrow \mathbb{O}^n$ inducing chain maps

$$C(\mathcal{T}^n) \rightarrow C(\chi^K(\mathcal{T}^n))$$
$$C(\chi^K(\mathcal{T}^n)) \rightarrow C(\mathbb{O}^n).$$

Let

$$a\colon C(S^n) \rightarrow C(\mathbb{O}^n)$$

be the result of composing these maps.

The chain ∂S^n is a boundary in $C(S^n)$, and therefore $a(\partial S^n)$ must also be a boundary in $C(\mathbb{O}^n)$. We will exploit the symmetry properties of the subdivisions and maps used to construct a, together with the annulus structure of the reduced renaming output complex $\mathbb{O}^n$, to show that $a(\partial S^n)$ *cannot* be a boundary chain in $C(\mathbb{O}^n)$. As a result of this contradiction, we must conclude that there is no anonymous reduced renaming protocol.

Because $\mathcal{T}^n$ is symmetric on its boundary, and because μ is symmetric under permutation, the chain maps induced by the permutation operators π_i^n, for $0 \leq i \leq n$, satisfy the following commutative property: For every chain κ of $C(face_n(S^n))$, $\pi_i^n(a(\kappa)) = a(\pi_i^n(\kappa))$. The conventional way to represent such relations is by a *commutative diagram*:

$$\begin{array}{ccc} C(face_n(S^n)) & \stackrel{a}{\rightarrow} & C(\mathbb{O}_n^{n-1}) \\ \downarrow {\scriptstyle \pi_i^n} & & \downarrow {\scriptstyle \pi_i^n} \\ C(face_i(S^n)) & \stackrel{a}{\rightarrow} & C(\mathbb{O}_i^{n-1}). \end{array}$$

Let $\mathcal{S}^{n-1}$ be the boundary complex of S^n, the complex consisting of all proper faces of S^n. The color-preserving isomorphism $\mathcal{S}^{n-1} \rightarrow \hat{0}^{n-1}$ induces a chain map

$$z\colon C(\mathcal{S}^{n-1}) \rightarrow C(\mathbb{O}^n).$$

Note that $\partial 0^n = z(\partial S^n)$. This map also commutes with π_i^n:

$$\begin{array}{ccc} C(face_n(\mathcal{S}^n)) & \stackrel{z}{\rightarrow} & C(\mathbb{O}_n^{n-1}) \\ \downarrow {\scriptstyle \pi_i^n} & & \downarrow {\scriptstyle \pi_i^n} \\ C(face_i(\mathcal{S}^n)) & \stackrel{z}{\rightarrow} & C(\mathbb{O}_i^{n-1}). \end{array}$$

LEMMA 6.12. *For $0 \leq i < n$, there is a chain map*

$$d\colon C_i(S^n) \rightarrow C_{i+1}(\mathbb{O}^n)$$

such that for every face S^m of S^n, $0 \leq m < n$

$$a(S^m) - z(S^m) - d(\partial S^m)$$

is an m-cycle of $\mathbb{O}^n$. Moreover, for every $0 \leq m < n$, the following diagram commutes:

$$\begin{array}{ccc} C(face_n(\mathcal{S}^n)) & \xrightarrow{d} & C(\mathbb{O}_n^{n-1}) \\ \downarrow \pi_i^n & & \downarrow \pi_i^n \\ C(face_i(\mathcal{S}^n)) & \xrightarrow{d} & C(\mathbb{O}_i^{n-1}). \end{array}$$

PROOF. We argue by induction on m. When $m = 1$, $ids(S^1) = \{i, j\}$. The 1-chains $a(S^1)$ and $z(S^1)$ have the same boundary: $\langle P_i, 0\rangle - \langle P_j, 0\rangle$, so $a(S^1) - z(S^1)$ is a cycle, and $d(\langle P_i, 0\rangle) = 0$.

Assume the claim for m, $1 \leq m < n - 1$. By Lemma 6.10, every m-cycle of $\mathbb{O}^n$ is a boundary (for $m < n - 1$), so there exists an $(m + 1)$-chain $d(S^m)$ such that

$$\partial d(S^m) = a(S^m) - z(S^m) - d(\partial S^m).$$

Because $S^m = face_{m+1}(S^{m+1})$,

$$\begin{aligned} &\partial d(face_{m+1}(S^{m+1})) \\ &\quad = a(face_{m+1}(S^{m+1})) - z(face_{m+1}(S^{m+1})) - d(\partial face_{m+1}(S^{m+1})). \end{aligned}$$

Applying π_i^n to both sides,

$$\begin{aligned} &\pi_i^n(\partial d(face_{m+1}(S^{m+1}))) \\ &\quad = \pi_i^n(a(face_{m+1}(S^{m+1})) - z(face_{m+1}(S^{m+1})) - d(\partial face_{m+1}(S^{m+1}))). \end{aligned}$$

By the induction hypothesis, π_i^n commutes with d, a, and z, and it commutes with the boundary operator because it is a chain map.

$$\begin{aligned} &\partial d(\pi_i^n(face_{m+1}(S^{m+1}))) \\ &\quad = a(\pi_i^n(face_{m+1}(S^{m+1})) - z(\pi_i^n(face_{m+1}(S^{m+1}))) \\ &\qquad - d(\partial \pi_i^n(face_{m+1}(S^{m+1}))). \end{aligned}$$

Because $\pi_i^n(face_{m+1}(S^{m+1})) = face_i(S^{m+1}))$,

$$\partial d(face_i(S^{m+1})) = a(face_i(S^{m+1}) - z(face_i(S^{m+1})) - d(\partial face_i(S^{m+1})).$$

Taking the alternating sum over the faces of S^{m+1} yields

$$\begin{aligned} \partial d(\partial S^{m+1}) &= a(\partial S^{m+1}) - z(\partial S^{m+1}) - d(\partial \partial S^{m+1}) \\ &= a(\partial S^{m+1}) - z(\partial S^{m+1}). \end{aligned}$$

Rearranging terms yields

$$0 = \partial(a(S^{m+1}) - z(S^{m+1}) - d(\partial S^{m+1})),$$

implying that

$$C = a(S^{m+1}) - z(S^{m+1}) - d(\partial S^{m+1})$$

is an $(m + 1)$-cycle. □

THEOREM 6.13. *There is no wait-free reduced renaming protocol.*

PROOF. As noted above, it suffices to prove that the chain $a(\partial S^n)$ is not a boundary in $C(\mathbb{O}^n)$. By Lemma 6.12,

$$a(face_n(S^{n-1})) - z(face_n(S^{n-1})) - d(\partial(face_n(S^{n-1})))$$

is an $(n - 1)$-cycle of $\mathbb{O}^n$. Lemma 6.10 implies that this cycle is homologous to $k \cdot \partial B_n$, for some integer k. Recall that $\pi_i^n(B_n) = B_i$.

$$\pi_i^n(a(face_n(S^n)) - z(face_n(S^n)) - d(\partial(face_n(S^n)))) \sim k \cdot \pi_i^n(B_n) = k \cdot B_i.$$

Recall that $\pi_i^n(face_n(S^n)) = face_i(S_n)$. Taking the alternating sum over the $(n - 1)$-dimensional faces of S^n yields:

$$a(\partial S^n) - z(\partial S^n) - d(\partial\partial S^n) \sim k \sum_{i=0}^{n} (-1)^i B_i$$

Because $\partial\partial S^n = 0$, $z(\partial S^n) = 0^n$, and $B_i \sim (-1)^i \partial 0^n$ (Lemma 6.11),

$$a(\partial S^n) \sim (1 + (n + 1)k) \cdot \partial 0^n.$$

Since there is no value of k for which $(1 + (n + 1)k)$ is zero, the cycle $a(\partial S^n)$ is not a boundary, yielding the desired contradiction. □

COROLLARY 6.14. *There is no wait-free $(n + 1)$-process read/write renaming protocol with $2n$ or fewer names.*

7. *Discussion*

Since the conference version of this paper appeared, our topological model has yielded a variety of additional results. Herlihy and Rajsbaum [1994] consider protocol complexes for protocols that employ more powerful primitives than read/write memory. Chaudhuri et al. [1993] give the first tight topology-based lower bounds on set agreement in the synchronous fail-stop model. Attiya and Rajsbaum [1995] cast our topological model in an equivalent "combinatorial" representation. Borowsky [1995] and Borowsky and Gafni [1993] base a key part of their simulation method on our notion of spans. Herlihy and Rajsbaum [1994] use homology theory to derive further impossibility results for set agreement, and to unify a variety of known impossibility results in terms of the algebraic theory of chain maps and chain complexes [Herlihy and Rajsbaum 1997]. The impossibility proof for renaming reflects the influence of this paper.

The graph theoretic characterization of Biran et al. [1988] also provides an effective procedure for deciding whether a task has a 1-resilient message passing protocol. By contrast, Gafni and Koutsoupias [1996] use topological techniques to show that it is undecidable in general whether wait-free read-write tasks have

a read-write protocol. Herlihy and Rajsbaum [1997] extend these techniques to characterize task decidability in a variety of computational models. Herlihy et al. [1998] give a simple round-by-round construction that unifies the synchronous, semi-synchronous, and asynchronous message-passing models of distributed computation within a common formalism based on a topological construction called a *pseudosphere*.

Herlihy and Rajsbaum [1998] use topological techniques to give the first complete characterization of the computational power of a nontrivial family of synchronization primitives, encompassing both read-write memory and three-process set agreement.

The topological framework is also of use in modeling complexity. Hoest and Shavit [1997] analyze the round complexity of protocols in the iterated immediate snapshot (IIS) model of Borowsky and Gafni. By introducing a novel form of span called the *nonuniform chromatic subdivision*, they refine our topological computability model into a theorem that states that the time complexity of any IIS protocol is directly proportional to the level of nonuniform chromatic subdivisions necessary to allow a simplicial map from a task's input complex to its output complex. In other words, the more you need to subdivide in order for a map to exist, the higher the complexity of your algorithm.

We believe the topological approach has a great deal of promise for the theory of distributed and concurrent computation, and that it merits further investigation. We look forward to the day when knowledge of elementary combinatorial and algebraic topology is considered as essential to theoretical computer science as knowledge of graph theory or probability theory.

Appendix A

A1. Connectivity

Theorem A1. *If $\mathcal{A}$ and $\mathcal{B}$ are each n-connected, and $\mathcal{A} \cap \mathcal{B}$ is $(n-1)$-connected, then $\mathcal{A} \cup \mathcal{B}$ is n-connected.*

Proof. Recall that a complex $\mathcal{C}$ is n-connected if its homotopy groups $\pi_1(C), \ldots, \pi_n(\mathcal{C})$ vanish [Spanier 1966].

By induction on n. For the base case, when $n = 1$, this theorem is just the Siefert/Van Kampen theorem [Spanier 1966, p. 151]. For the induction step, assume $\mathcal{A}$ and $\mathcal{B}$ are n-connected, and $\mathcal{A} \cap \mathcal{B}$ is $(n-1)$-connected, and $\mathcal{A} \cup \mathcal{B}$ is $(n-1)$-connected. The homology groups $H_n(\mathcal{A})$, $H_n(\mathcal{B})$, and $H_{n-1}(\mathcal{A} \cap \mathcal{B})$ all vanish. and by the Mayer-Vietoris sequence [Spanier 1966, p. 186], so does the homology group $H_n(\mathcal{A} \cup \mathcal{B})$. By the Hurewicz Isomorphism Theorem [Spanier 1966, p. 394], $\pi_n(\mathcal{A} \cup \mathcal{B})$ must also vanish and therefore $\mathcal{A} \cup \mathcal{B}$ is n-connected. □

A2. Anonymous Computability Theorem

Definition A2. The *chromatic K-extension* of a color-preserving simplicial map $\phi: \mathcal{A} \to \mathcal{B}$ is the map $\psi: \chi^K(\mathcal{A}) \to \mathcal{B}$ defined by $\psi(\vec{x}) = \phi(\vec{y})$, where $\vec{y}$ is the unique vertex in carrier$(\vec{x}, \mathcal{A})$ of matching color.

Lemma A3. *The chromatic K-extension of any color-preserving simplicial map is a color-preserving simplicial map.*

PROOF. Define ξ: $\chi^K(\mathcal{A}) \rightarrow \mathcal{A}$ to be the map carrying each $\vec{x}$ to $\vec{y}$, the unique vertex in *carrier*($\vec{x}$, $\mathcal{A}$) of matching color. By Lemma 5.27, ξ is a simplicial map. The chromatic K-extension of ϕ is the composition of ϕ and ξ, both color-preserving simplicial maps. □

LEMMA A4. *Let $\mathcal{A}$ be a $(p-1)$-sphere, $\mathcal{B}$ a p-disk having $\mathcal{A}$ as boundary, and $\mathcal{C}$ a complex that is $(p-1)$-connected and link-connected. If ϕ: $\mathcal{A} \rightarrow \mathcal{C}$ is a color-preserving simplicial map, then there exists a color-preserving simplicial map*

$$\psi\colon \chi^K(\mathcal{B}) \rightarrow \mathcal{A}$$

for some $K \geq 0$, such that ψ restricted to $\chi^K(\mathcal{A})$ is the K-chromatic extension of ϕ.

PROOF. Lemma 4.21 states that there exists a chromatic subdivision σ and simplicial map

$$\hat{\phi}\colon \sigma(\mathcal{B}) \rightarrow \mathcal{C}$$

such that $\sigma(\mathcal{A}) = \mathcal{A}$, and $\hat{\phi}$ agrees with ϕ on $\mathcal{A}$.

By Theorem 5.29, there exists $K \geq 0$ and a color and carrier-preserving simplicial map

$$\gamma\colon \chi^K(\mathcal{B}) \rightarrow \sigma(\mathcal{B}).$$

The composition of $\hat{\phi}$ and γ yields the desired map ψ. □

An *anonymous span* is a color-preserving map ϕ: $\chi^K(\mathcal{I}) \rightarrow \mathcal{P}$, for some $K \geq 0$, such that ϕ is symmetric under permutation, and for all X in $\chi^K(\mathcal{I})$,

$$\phi(X) \in \mathcal{P}(\mathit{carrier}(X, \sigma(\mathcal{I}))). \tag{14}$$

Definition A5. Let $\mathcal{C}$ be a symmetric complex. Two k-simplexes S_0^k and S_1^k belong to the same *k-orbit* if $S_0^k = \pi(S_1^k)$, for some permutation π.

The set of k-orbits partition the k-simplexes of $\mathcal{C}$ into equivalence classes.

LEMMA A6. *Every wait-free anonymous protocol complex has an anonymous span.*

PROOF. We build up the span inductively. For each $\vec{v} \in \mathcal{I}$, define $\phi_0(\vec{v}) = \mathcal{P}(\vec{v})$, the unique vertex corresponding to a solo execution. This map trivially satisfies Eq. 14, and is symmetric under permutation.

Assume inductively that for some $K_{k-1} \geq 0$, we have a color-preserving simplicial map

$$\phi_{k-1}\colon \chi^{K_{k-1}}(\mathit{skel}^{k-1}(\mathcal{I})) \rightarrow \mathcal{P}$$

symmetric under permutation, and satisfying Eq. (14). Let $S_0^k, \ldots, S_L^k$ be a set of k-simplexes constructed by choosing one k-simplex from each k-orbit of $\mathit{skel}^k(\mathcal{I})$.

For $0 \leq i \leq L$, $\chi^{K_{k-1}}(\mathit{skel}^{k-1}(S_i^k))$ is a $(k-1)$-sphere, and the subdivision of S_i^k constructed by starring $\chi^{K_{k-1}}$ is a k-disk, so by Lemma A4, for some $L_i \geq 0$,

$$\phi_{k-1}\colon \chi^{K_{k-1}}(\dot{\mathcal{I}}_i^{k-1}) \rightarrow \mathcal{P}(S_i^k)$$

can be extended to a color-preserving simplicial map

$$\psi\colon \chi^{K_{k-1}+L_i}(S_i^k) \rightarrow \mathcal{P}(S_i^k),$$

such that ψ restricted to $\chi^{K_{k-1}+L_i}(skel^{k-1}(S_i^k))$ is the L_i-chromatic extension of ϕ_{k-1}. Let $K_k = \max_{i=0}^{L}(K_{k-1} + L_i)$. For $0 \leq i \leq L$, define

$$\psi_i\colon \chi^{K_k}(S_i^k) \rightarrow \mathcal{P}(S_i^k)$$

to be the L_i-chromatic extension of ψ. The restriction of ψ_i to $skel^{k-1}(S_i^k)$ is the $(K_k - K_{k-1})$-chromatic extension of ϕ_{k-1}, so for every $0 \leq i, j \leq L$, ψ_i and ψ_j agree on the intersection of their domains. Together they define a map

$$\phi_k\colon \chi^{K_k}(skel^{k-1}(\mathcal{I})) \rightarrow \mathcal{P}$$

satisfying Eq. 14. This completes the induction step of the proof. □

THEOREM A7 (ANONYMOUS COMPUTABILITY THEOREM). *A symmetric decision task* $\langle \mathcal{I}^n, \mathcal{O}^n, \Delta \rangle$ *has a wait-free anonymous protocol using read-write memory if and only if there exists an integer K and a color-preserving simplicial map*

$$\mu\colon \chi^K(\mathcal{I}) \rightarrow \mathcal{O}$$

symmetric under permutation, such that for each simplex X in $\chi^K(\mathcal{I})$, $\mu(X) \in \Delta(carrier(X, \mathcal{I}))$.

PROOF. The "if" part follows immediately from the protocol construction in Section 5.

The "only if" part follows from the existence of the anonymous span guaranteed by Lemma A6. □

ACKNOWLEDGMENTS. We wish to thank the many people whose comments have helped to improve this paper over the past six years: Yehuda Afek, Hagit Attiya, Elizabeth Borowsky, Faith Fich, Gunnar Hoest, Alan Fekete, Eli Gafni, Nancy Lynch and her students, Shlomo Moran, Lyle Ramshaw, Eric Ruppert, Mark Tuttle, Gideon Stupp, and most of all, Sergio Rajsbaum and John Havlicek.

REFERENCES

AFEK, Y., ATTIYA, H., DOLEV, D., GAFNI, E., MERRITT, M., AND SHAVIT, N. 1990. Atomic snapshots of shared memory. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing* (Quebec City, Que., Canada, Aug. 22–24). ACM, New York, pp. 1–14.

AFEK, Y., AND STUPP, G. 1993. Synchronization power depends on the register size (preliminary version). In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, Los Alamitos, Calif. 195–205.

ANDERSON, J. 1990. Composite registers. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing* (Quebec City, Que., Canada, Aug. 22–24). ACM, New York, pp. 15–30.

ATTIYA, H., BAR-NOY, A., DOLEV, D., PELEG, D., AND REISCHUK, R. 1990. Renaming in an asynchronous environment. *J. ACM*, July.

ATTIYA, H., LYNCH, N., AND SHAVIT, N. 1990. Are wait-free algorithms fast? In *Proceedings of the 31st Annual Symposium on the Foundations of Computer Science.* IEEE Computer Society Press, Los Alamitos, Calif. 55–64.

ATTIYA, H., AND RAJSBAUM, S. 1995. A combinatorial topology framework for wait-free computability. Preprint.

ATTIYA, H., AND WELCH, J. 1998. *Distributed Computing: fundamentals, simulations and advanced topics.* McGraw-Hill, London, England. ISBN 0-07-7093526.

BAR-NOY, A., AND DOLEV, D. 1989. Shared-memory vs. message-passing in an asynchronous distributed environment. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing* (Edmonton, Alb., Canada, Aug. 14–16). ACM, New York, pp. 307–318.

BIRAN, O., MORAN, S., AND ZAKS, S. 1988. A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing* (Toronto, Ont., Canada, Aug. 15–17). ACM, New York, pp. 263–275.

BOROWSKY, E. 1995. Capturing the power of resiliency and set consensus in distributed systems. Tech. rep., University of California Los Angeles, Los Angeles, Calif.

BOROWSKY, E., AND GAFNI, E. 1993. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing* (San Diego, Calif., May 16–18). ACM, New York, pp. 91–100.

BOROWSKY, E., AND GAFNI, E. 1993. Immediate atomic snapshots and fast renaming. In *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing* (Ithaca, N.Y., Aug. 15–18). ACM, New York, pp. 41–52.

CHAUDHURI, S. 1990. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing* (Quebec City, Que., Canada, Aug. 22–24). ACM, New York, pp. 311–324.

CHAUDHURI, S., HERLIHY, M. P., LYNCH, N., AND TUTTLE, M. R. 1993. A tight lower bound for k-set agreement. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science* (Oct.). IEEE Computer Society Press, Los Alamitos, Calif., pp. 206–215.

CHOR, B., ISRAELI, A., AND LI, M. 1987. On processor coordination using asynchronous hardware. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing* (Vancouver, B.C., Canada, Aug. 10–12). ACM, New York, pp. 86–97.

CHOR, B., AND MOSCOVICI, L. 1989. Solvability in asynchronous environments. In *IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 422–427.

DOLEV, D., DWORK, C., AND STOCKMEYER, L. 1987. On the minimal synchronism needed for distributed consensus. *J. ACM 34*, 1 (Jan.), 77–97.

FEKETE, A. 1986. Asymptotically optimal algorithms for approximate agreement. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing* (Calgary, Alb., Canada, Aug. 11–13). ACM, New York, pp. 73–87.

FISCHER, M., LYNCH, N. A., AND PATERSON, M. S. 1985. Impossibility of distributed commit with one faulty process. *J. ACM 32*, 2 (Apr.).

GAFNI, E., AND KOUTSOUPIAS, E. 1996. Three-processor tasks are undecidable. http://daphne.cs.ucla.edu/eli/undec.ps.

GLASER, L. C. 1970. *Geometrical Combinatorial Topology*, Vol. 1. Van Nostrand Reinhold, New York.

HERLIHY, M. P. 1991. Wait-free synchronization. *ACM Trans. Prog. Lang. Syst. 13*, 1 (Jan.), 123–149.

HERLIHY, M. P., AND RAJSBAUM, S. 1994. Set consensus using arbitrary objects. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing* (Los Angeles, Calif., Aug. 14–17). ACM, New York, pp. 324–333.

HERLIHY, M., AND RAJSBAUM, S. 1995. Algebraic spans. In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing* (Ottawa, Ont., Canada, Aug. 20–23). ACM, New York, pp. 90–99.

HERLIHY, M. P., AND RAJSBAUM, S. 1997. The decidability of distributed decision tasks. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (El Paso, Tex., May 4–6). ACM, New York, pp. 589–598.

HERLIHY, M., AND RAJSBAUM, S. 1998. A wait-free classification of loop agreement tasks. In *Proceedings of the 12th International Symposium on Distributed Computing* (Sept.). Springer-Verlag, New York, pp. 175–185.

HERLIHY, M. P., RAJSBAUM, S., AND TUTTLE, M. R. 1998. Unifying synchronous and asynchronous message-passing models. In *Proceedings of the 12th International Symposium on Distributed Computing* (Sept.).

HERLIHY, M. P., AND SHAVIT, N. 1993. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing* (San Diego, Calif., May 16–18). ACM, New York, pp. 111–120.

HERLIHY, M. P., AND SHAVIT, N. 1994. A simple constructive computability theorem for wait-free computation. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing* (Montreal, Que., Canada, May 23–25). ACM, New York, pp. 243–252.

HERLIHY, M. P., AND WING, J. M. 1990. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Prog. Lang. Syst. 12*, 3 (July), 463–492.

HOEST, G. 1997. Towards a Topological Characterization of Asynchronous Complexity. Ph.D. dissertation. Mass. Institute of Technology, Cambridge, Mass.

HOEST, G., AND SHAVIT, N. 1997. Towards a topological characterization of asynchronous complexity. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing* (Santa Barbara, Calif., Aug. 21–24). ACM, New York, pp. 199–208.

LEFSCHETZ, S. 1949. *Introduction to Topology*. Princeton University Press, Princeton, N.J.

LOUI, M. C., AND ABU-AMARA, H. H. 1987. *Memory Requirements for Agreement Among Unreliable Asynchronous Processes*, vol. 4. JAI Press, Greenwich, Conn., pp. 163–183.

LYNCH, N. A. 1996. *Distributed Algorithms*. Morgan-Kaufmann, New York.

LYNCH, N. A., AND TUTTLE, M. R. 1988. An introduction to input/output automata. Tech. Rep. MIT/LCS/TM-373. MIT Laboratory for Computer Science, Cambridge, Mass.

MUNKRES, J. R. 1984. *Elements of Algebraic Topology*. Addison-Wesley, Reading, Mass. ISBN 0-201-04586-9.

SAKS, M., AND ZAHAROGLOU, F. 1993. Wait-free k-set agreement is impossible: The topology of public knowledge. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing* (San Diego, Calif., May 16–18). ACM, New York, pp. 101–110.

SPANIER, E. H. 1966. *Algebraic Topology*. Springer-Verlag, New York.

RECEIVED FEBRUARY 1996; REVISED APRIL 1999; ACCEPTED MAY 1999

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

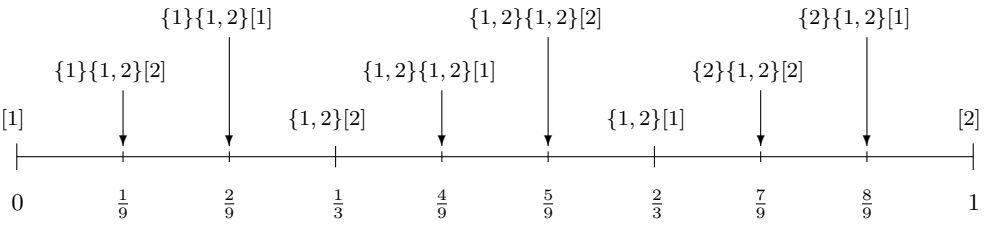

FIG. 1. *The map from* $\hat{\Sigma}(\{1,2\})$ *to the* $[0,1]$ *interval.*

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

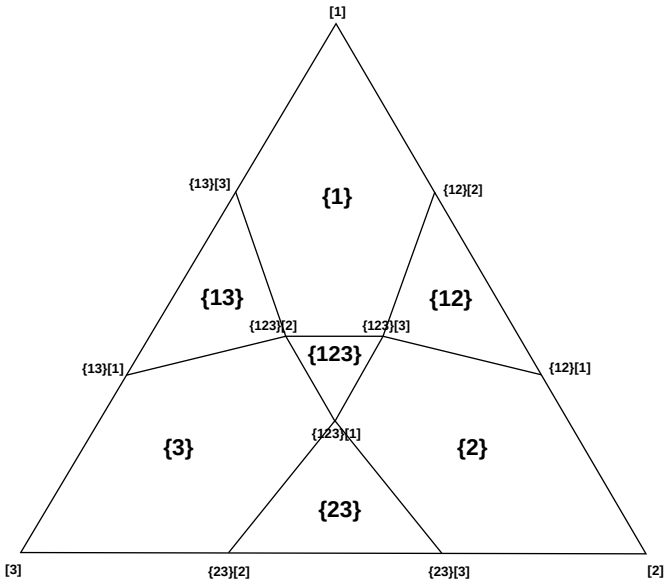

FIG. 2. *The subdivision* $D_0(\{1,2,3\})$.

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 $B(\vec{x},\epsilon)$.*

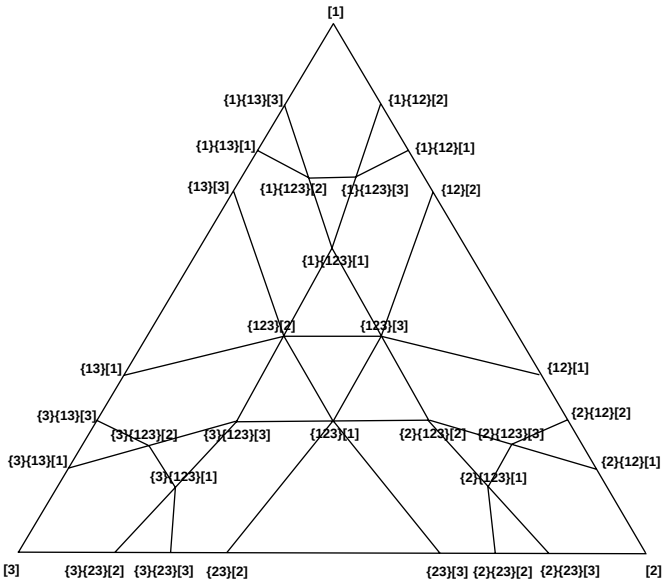

FIG. 3. *The subdivision* $D_1(\{1,2,3\})$ *with vertex labels.*

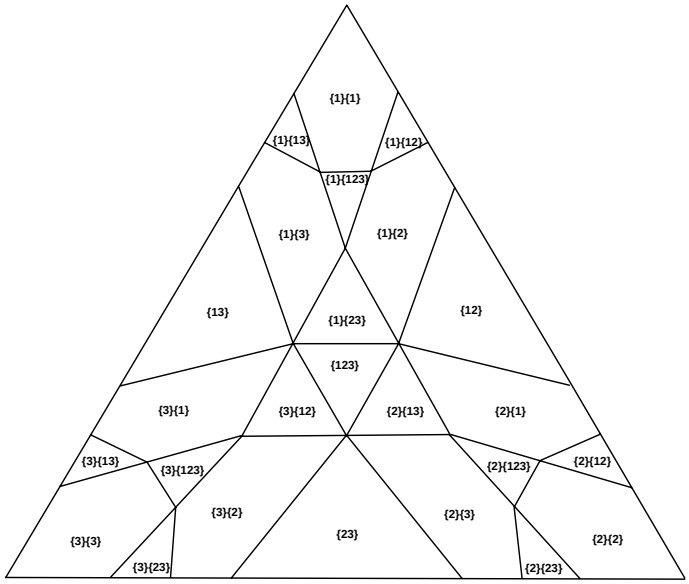

FIG. 4. *The subdivision* $D_1(\{1,2,3\})$ *with face labels.*

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 $\tau\{j,k\}[k]$.

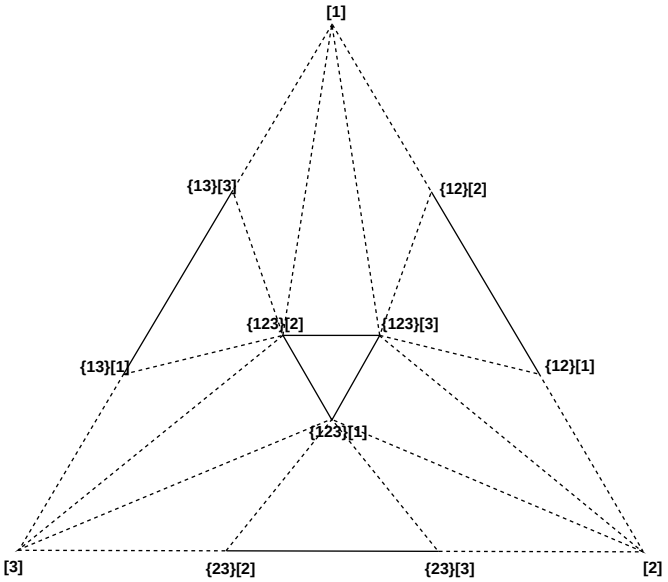

FIG. 5. *The graph* $G_0(\{1,2,3\})$.

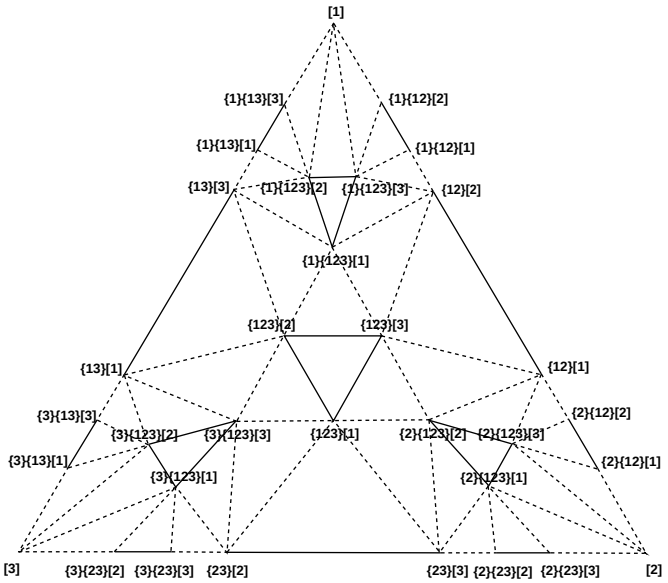

FIG. 6. *The graph* $G_1(\{1,2,3\})$.

(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:*

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

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.

Knowledge and Common Knowledge in a Distributed Environment

JOSEPH Y. HALPERN

IBM Almaden Research Center,
San Jose, California

AND

YORAM MOSES

The Weizmann Institute of Science,
Rehovot, Israel

Abstract. Reasoning about knowledge seems to play a fundamental role in distributed systems. Indeed, such reasoning is a central part of the informal intuitive arguments used in the design of distributed protocols. Communication in a distributed system can be viewed as the act of transforming the system's state of knowledge. This paper presents a general framework for formalizing and reasoning about knowledge in distributed systems. It is shown that states of knowledge of groups of processors are useful concepts for the design and analysis of distributed protocols. In particular, *distributed knowledge* corresponds to knowledge that is "distributed" among the members of the group, while *common knowledge* corresponds to a fact being "publicly known." The relationship between common knowledge and a variety of desirable actions in a distributed system is illustrated. Furthermore, it is shown that, formally speaking, in practical systems common knowledge cannot be attained. A number of weaker variants of common knowledge that are attainable in many cases of interest are introduced and investigated.

Categories and Subject Descriptors: B.4.4 **[Input/Output and Data Communications]**: Performance Analysis and Design Aids—*formal models, verificiation*; B.4.5 **[Input/Output and Data Communications]**: Reliability, Testing, and Fault-Tolerance; C.2.2 **[Computer-Communication Networks]**: Network Protocols—*protocol verification*; D.2.4 **[Software Engineering]**: Program Verification; D.2.10 **[Software Engineering]**: Design—*methodologies*; F.3.1 **[Logics and Meanings of Programs]**: Specifying and Verifying and Reasoning about Programs; I.2.4 **[Artificial Intelligence]**: Knowledge Representation Formalisms and Methods

General Terms: Algorithms, Reliability, Theory, Verification

Additional Key Words and Phrases: Common knowledge, distributed knowledge, knowledge, knowledge and action, variants of common knowledge

---

This is a revised and expanded version of a paper with the same title that first appeared in the *Proceedings of the 3rd ACM Conference on Principles of Distributed Computing* (Vancouver, B.C., Canada, Aug. 27–29). ACM, New York, 1984, pp. 50–61.

The work of Y. Moses was supported in part by DARPA contract N00039-82-C-0250.

Authors' addresses: J. Y. Halpern, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120-6099; Y. Moses, Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel.

1. *Introduction*

Distributed systems of computers are rapidly gaining popularity in a wide variety of applications. However, the distributed nature of control and information in such systems makes the design and analysis of distributed protocols and plans a complex task. In fact, at the current time, these tasks are more an art than a science. Basic foundations, general techniques, and a clear methodology are needed to improve our understanding and ability to deal effectively with distributed systems.

Although the tasks that distributed systems are required to perform are normally stated in terms of the global behavior of the system, the actions that a processor performs can depend only on its local information. Since the design of a distributed protocol involves determining the behavior and interaction between individual processors in the system, designers frequently find it useful to reason intuitively about processors' "states of knowledge" at various points in the execution of a protocol. For example, it is customary to argue that ". . . once the sender receives the acknowledgment, it *knows* that the current packet has been delivered; it can then safely discard the current packet, and send the next packet . . . ". Ironically, however, formal descriptions of distributed protocols, as well as actual proofs of their correctness or impossibility, have traditionally avoided any explicit mention of knowledge. Rather, the intuitive arguments about the state of knowledge of components of the system are customarily buried in combinatorial proofs that are unintuitive and hard to follow.

The general concept of knowledge has received considerable attention in a variety of fields, ranging from Philosophy [25] and Artificial Intelligence [32, 33] to Game Theory [1] and Psychology [5]. The main purpose of this paper is to demonstrate the relevance of reasoning about knowledge to distributed systems as well. Our basic thesis is that explicitly reasoning about the states of knowledge of the components of a distributed system provides a more general and uniform setting that offers insight into the basic structure and limitations of protocols in a given system.

As mentioned above, agents can only base their actions on their local information. This knowledge, in turn, depends on the messages they receive and the events they observe. Thus, there is a close relationship between knowledge and action in a distributed environment. When we consider the task of performing coordinated actions among a number of agents in a distributed environment, it does not, in general, suffice to talk only about individual agent's knowledge. Rather, we need to look at states of knowledge of *groups* of agents (the group of all participating agents is often the most relevant one to consider). Attaining particular states of group knowledge is a prerequisite for performing coordinated actions of various kinds.

In this work we define a hierarchy of states of group knowledge. It is natural to think of communication in the system as the act of improving the state of knowledge, in the sense of "climbing up the hierarchy." The weakest state of knowledge we discuss is *distributed knowledge*, which corresponds to knowledge that is distributed among the members of the group, without any individual agent necessarily having it.[1] The strongest state of knowledge in the hierarchy is *common knowledge*, which roughly corresponds to "public knowledge." We show that the execution of simultaneous actions becomes common knowledge, and hence that

[1] In a previous version of this paper [22], what we are now calling distributed knowledge was called implicit knowledge. We have changed the name here to avoid conflict with the usage of the phrase "implicit knowledge" in papers such as [11] and [29].

such actions cannot be performed if common knowledge cannot be attained. Reaching agreement is an important example of a desirable simultaneous action in a distributed environment. A large part of the technical analysis in this paper is concerned with the ability and cost of attaining common knowledge in systems of various types. It turns out that attaining common knowledge in distributed environments is not a simple task. We show that when communication is not guaranteed it is impossible to attain common knowledge. This generalizes the impossibility of a solution to the well-known *coordinated attack* problem [17]. A more careful analysis shows that common knowledge can only be attained in systems that support simultaneous coordinated actions. It can be shown that such actions cannot be guaranteed or detected in practical distributed systems. It follows that common knowledge cannot be attained in many cases of interest. We then consider states of knowledge that correspond to eventually coordinated actions and to coordinated actions that are guaranteed to be performed within a bounded amount of time. These are essentially weaker variants of common knowledge. However, whereas, strictly speaking, common knowledge cannot be attained in many practical cases, these weaker states of knowledge often can be attained.

Another question that we consider is that of when it is safe to assume that certain facts are common knowledge, even when strictly speaking they are not. For this purpose, we introduce the concept of *internal knowledge consistency*. Roughly speaking, it is internally knowledge consistent to assume that a certain state of knowledge holds at a given point, if nothing the processors in the system will ever encounter will be inconsistent with this assumption.

The rest of the paper is organized as follows: In the next section, we look at the "muddy children" puzzle, which illustrates some of the subtleties involved in reasoning about knowledge in the context of a group of agents. In Section 3, we introduce a hierarchy of states of knowledge in which a group may be. Section 4 focuses on the relationship between knowledge and communication by looking at the coordinated attack problem. In Section 5, we sketch a general definition of a distributed system, and, in Section 6, we discuss how knowledge can be ascribed to processors in such systems so as to make statements such as "agent 1 *knows* φ" completely formal and precise. Section 7 relates common knowledge to the coordinated attack problem. In Section 8, we show that, strictly speaking, common knowledge cannot be attained in practical distributed systems. Section 9 considers the implications of this observation and in Section 10 we begin to reconsider the notion of common knowledge in the light of these implications. In Sections 11 and 12, we consider a number of variants of common knowledge that are attainable in many cases of interest and discuss the relevance of these states of knowledge to the actions that can be performed in a distributed system. Section 13 discusses the notion of internal knowledge consistency, and Section 14 contains some concluding remarks.

2. *The Muddy Children Puzzle*

A crucial aspect of distributed protocols is the fact that a number of different processors cooperate in order to achieve a particular goal. In such cases, since more than one agent is present, an agent may have knowledge about other agents' knowledge in addition to his knowledge about the physical world. This often requires care in distinguishing subtle differences between seemingly similar states of knowledge. A classical example of this phenomenon is the muddy children puzzle—a variant of the well known "wise men" or "cheating wives" puzzles. The

version given here is taken from [2]:

> Imagine n children playing together. The mother of these children has told them that if they get dirty there will be severe consequences. So, of course, each child wants to keep clean, but each would love to see the others get dirty. Now it happens during their play that some of the children, say k of them, get mud on their foreheads. Each can see the mud on others but not on his own forehead. So, of course, no one says a thing. Along comes the father, who says, "At least one of you has mud on your head," thus expressing a fact known to each of them before he spoke (if $k > 1$). The father then asks the following question, over and over: "Can any of you prove you have mud on your head?" Assuming that all the children are perceptive, intelligent, truthful, and that they answer simultaneously, what will happen?

The reader may want to think about the situation before reading the rest of Barwise's discussion:

> There is a "proof" that the first $k - 1$ times he asks the question, they will all say "no" but then the kth time the dirty children will answer "yes."
>
> The "proof" is by induction on k. For $k = 1$, the result is obvious: the dirty child sees that no one else is muddy, so he must be the muddy one. Let us do $k = 2$. So there are just two dirty children, a and b. Each answers "no" the first time, because of the mud on the other. But, when b says "no," a realizes that he must be muddy, for otherwise b would have known the mud was on his head and answered "yes" the first time. Thus, a answers "yes" the second time. But b goes through the same reasoning. Now suppose $k = 3$; so there are three dirty children, a, b, c. Child a argues as follows. Assume I don't have mud on my head. Then, by the $k = 2$ case, both b and c will answer "yes" the second time. When they don't, he realizes that the assumption was false, that he is muddy, and so will answer "yes" on the third question. Similarly for b and c. [The general case is similar.]

Let us denote the fact "At least one child has a muddy forehead" by **m**. Notice that if $k > 1$, that is, more than one child has a muddy forehead, then every child can see at least one muddy forehead, and the children initially all know **m**. Thus, it would seem, the father does not need to tell the children that **m** holds when $k > 1$. But this is false! In fact, had the father not announced **m**, the muddy children would never have been able to conclude that their foreheads are muddy. We now sketch a proof of this fact.

First of all, given that the children are intelligent and truthful, a child with a clean forehead will never answer "yes" to any of the father's questions. Thus, if $k = 0$, all of the children answer all of the father's questions "no." Assume inductively that if there are exactly k muddy children and the father does not announce **m**, then the children all answer "no" to all of the father's questions. Note that, in particular, when there are exactly k muddy foreheads, a child with a clean forehead initially sees k muddy foreheads and hears all of the father's questions answered "no." Now assume that there are exactly $k + 1$ muddy children. Let $q \geq 1$ and assume that all of the children answer "no" to the father's first $q - 1$ questions. We have argued above that a clean child will necessarily answer "no" to the father's qth question. Next observe that before answering the father's qth question, a muddy child has exactly the same information as a clean child has at the corresponding point in the case of k muddy foreheads. It follows that the

muddy children must all answer "no" to the father's qth question, and we are done. (A very similar proof shows that if there are k muddy children and the father does announce **m**, his first $k-1$ questions are answered "no.")

So, by announcing something that the children all know, the father somehow manages to give the children useful information! How can this be? Exactly what *was* the role of the father's statement? In order to answer this question, we need to take a closer look at knowledge in the presence of more than one knower; this is the subject of the next section.

3. *A Hierarchy of States of Knowledge*

In order to analyze the muddy children puzzle introduced in the previous section, we need to consider states of knowledge of groups of agents. As we shall see in the sequel, reasoning about such states of knowledge is crucial in the context of distributed systems as well. In Section 6, we shall carefully define what it means for an agent i to know a given fact φ (which we denote by $K_i\varphi$). For now, however, we need knowledge to satisfy only two properties. The first is that an agent's knowledge at a given time must depend only on its local history: the information that it started out with combined with the events it has observed since then. Secondly, we require that only true things be known, or more formally:

$$K_i\varphi \supset \varphi;$$

that is, if an agent i knows φ, then φ is true. This property, which is occasionally referred to as the *knowledge axiom*, is the main property that philosophers customarily use to distinguish knowledge from belief (cf. [23]).

Given a reasonable interpretation for what it means for an agent to know a fact φ, how does the notion of knowledge generalize from an agent to a group? In other words, what does it mean to say that a group G of agents knows a fact φ? We believe that more than one possibility is reasonable, with the appropriate choice depending on the application:

—$D_G\varphi$ (read "the group G has *distributed knowledge* of φ"): We say that knowledge of φ is distributed in G if someone who knew everything that each member of G knows would know φ. For instance, if one member of G knows ψ and another knows that $\psi \supset \varphi$, the group G may be said to have distributed knowledge of φ.

—$S_G\varphi$ (read "*someone in G knows* φ"): We say that $S_G\varphi$ holds iff some member of G knows φ. More formally,

$$S_G\varphi \equiv \bigvee_{i\in G} K_i\varphi.$$

—$E_G\varphi$ (read "*everyone in G knows* φ"): We say that $E_G\varphi$ holds iff all members of G know φ. More formally,

$$E_G\varphi \equiv \bigwedge_{i\in G} K_i\varphi.$$

—$E_G^k\varphi$, for $k \geq 1$ (read "φ is *E^k-knowledge* in G"): $E_G^k\varphi$ is defined by

$$E_G^1\varphi = E_G\varphi,$$
$$E_G^{k+1}\varphi = E_G E_G^k\varphi, \qquad \text{for} \quad k \geq 1.$$

φ is said to be E^k-knowledge in G if "everyone in G knows that everyone in G knows that ... that everyone in G knows that φ is true" holds, where the phrase "everyone in G knows that" appears in the sentence k times.

—$C_G\varphi$ (read "φ is *common knowledge* in G"): The formula φ is said to be common knowledge in G if φ is E_G^k-knowledge for all $k \geq 1$. In other words,

$$C_G\varphi \equiv E_G\varphi \wedge E_G^2\varphi \wedge \cdots \wedge E_G^m\varphi \wedge \cdots$$

(We omit the subscript G when the group G is understood from context.)

Clearly, the notions of group knowledge introduced above form a hierarchy, with

$$C\varphi \supset \cdots \supset E^{k+1}\varphi \supset \cdots \supset E\varphi \supset S\varphi \supset D\varphi \supset \varphi.$$

However, depending on the circumstances, these notions might not be distinct. For example, consider a model of parallel computation in which a collection of n processors share a common memory. If their knowledge is based on the contents of the common memory, then we arrive at a situation in which $C\varphi \equiv E^k\varphi \equiv E\varphi \equiv S\varphi \equiv D\varphi$. By way of contrast, in a distributed system in which n processors are connected via some communication network and each one of them has its own memory, the above hierarchy is strict. Moreover, in such a system, every two levels in the hierarchy can be separated by an actual task, in the sense that there will be an action for which one level in the hierarchy will suffice, but no lower level will. It is quite clear that this is the case with $E\varphi \supset S\varphi \supset D\varphi$, and, as we are about to show, the "muddy children" puzzle is an example of a situation in which $E^k\varphi$ suffices to perform a required action, but $E^{k-1}\varphi$ does not. In the next section, we present the coordinated attack problem, a problem for which $C\varphi$ suffices to perform a required action, but for no k does $E^k\varphi$ suffice.

Returning to the muddy children puzzle, let us consider the state of the children's knowledge of $\mathbf{m}$: "At least one forehead is muddy." Before the father speaks, $E^{k-1}\mathbf{m}$ holds, and $E^k\mathbf{m}$ does not. To see this, consider the case $k = 2$ and suppose that Alice and Bob are the only muddy children. Clearly everyone sees at least one muddy child, so $E\mathbf{m}$ holds. But the only muddy child that Alice sees is Bob, and, not knowing whether she is muddy, Alice considers it possible that Bob is the only muddy child. Alice therefore considers it possible that Bob sees no muddy child. Thus, although both Alice and Bob know $\mathbf{m}$ (i.e., $E\mathbf{m}$ holds), Alice does not know that Bob knows $\mathbf{m}$, and hence $E^2\mathbf{m}$ does not hold. A similar argument works for the general case. We leave it to the reader to check that when there are k muddy children, $E^k\mathbf{m}$ suffices to ensure that the muddy children will be able to prove their dirtiness, whereas $E^{k-1}\mathbf{m}$ does not. (For a more detailed analysis of this argument, and for a general treatment of variants of the muddy children puzzle, see [35].)

Thus, the role of the father's statement was to improve the children's state of knowledge of $\mathbf{m}$ from $E^{k-1}\mathbf{m}$ to $E^k\mathbf{m}$. In fact, the children have *common knowledge* of $\mathbf{m}$ after the father announces that $\mathbf{m}$ holds. Roughly speaking, the father's public announcement of $\mathbf{m}$ to the children as a group results in all the children knowing $\mathbf{m}$ and knowing that the father has publicly announced $\mathbf{m}$. Assuming that it is common knowledge that all of the children know anything the father announces publicly, it is easy to conclude that the father's announcement makes $\mathbf{m}$ common knowledge. Once the father announces $\mathbf{m}$, all of the children know both $\mathbf{m}$ and that the father has announced $\mathbf{m}$. Every child thus knows that all of the children know $\mathbf{m}$ and know that the father publicly announced $\mathbf{m}$, and so $E^2\mathbf{m}$ holds. It is similarly possible to show that, once the father announces $\mathbf{m}$, then $E^k\mathbf{m}$ holds for all k, so $C\mathbf{m}$ holds (see Section 10 for further discussion). Since, in particular, $E^k\mathbf{m}$ holds, the muddy children can succeed in proving their dirtiness.

The vast majority of the communication in a distributed system can also be viewed as the act of improving the state of knowledge (in the sense of "climbing up a hierarchy") of certain facts. This is an elaboration of the view of communication in a network as the act of "sharing knowledge." Taking this view, two notions come to mind. One is *fact discovery*—the act of changing the state of knowledge of a fact from being distributed knowledge to levels of explicit knowledge (usually S-, E-, or C-knowledge), and the other is *fact publication*—the act of changing the state of knowledge of a fact that is not common knowledge to common knowledge. An example of fact discovery is the detection of global properties of a system, such as deadlock. The system initially has distributed knowledge of the deadlock, and the detection algorithm improves this state to S-knowledge (see [3] for work related to fact discovery). An example of fact publication is the introduction of a new communication convention in a computer network. Here the initiator(s) of the convention wish to make the new convention common knowledge.

In the rest of the paper, we devote a considerable amount of attention to fact publication and common knowledge. As we shall show, common knowledge is inherent in a variety of notions of agreement, conventions, and coordinated action. Furthermore, having common knowledge of a large number of facts allows for more efficient communication. Since these are goals frequently sought in distributed computing, the problem of fact publication—how to attain common knowledge—becomes crucial. Common knowledge is also a basic notion in everyday communication between people. For example, shaking hands to seal an agreement signifies that the handshakers have common knowledge of the agreement. Also, it can be argued [5] that when we use a definite reference such as "the president" in a sentence, we assume common knowledge of who is being referred to.

In [5], Clark and Marshall present two basic ways in which a group can come to have common knowledge of a fact. One is by membership in a community, for example, the meaning of a red traffic light is common knowledge in the community of licensed drivers. The other is by being copresent with the occurrence of the fact, for example, the father's gathering the children and publicly announcing the existence of muddy foreheads made that fact common knowledge. Notice that if, instead, the father had taken each child aside (without the other children noticing) and told her or him about it privately, this information would have been of no help at all.

In the context of distributed systems, community membership corresponds to information that the processors are guaranteed to have by virtue of their presence in the system (e.g., information that is "inserted into" the processors before they enter the system). However, it is not obvious how to simulate copresence or "public" announcements using message passing in a distributed system. As we shall see, there are serious problems and unexpected subtleties involved in attempting to do so.

4. *The Coordinated Attack Problem*

To get a flavor of the issues involved in attaining common knowledge by simulating copresence in a distributed system, consider the coordinated attack problem, originally introduced by Gray [17]:

> Two divisions of an army are camped on two hilltops overlooking a common valley. In the valley awaits the enemy. It is clear that if both divisions attack the enemy simultaneously, they will win the battle; whereas if only one

> division attacks, it will be defeated. The divisions do not initially have plans for launching an attack on the enemy, and the commanding general of the first division wishes to coordinate a simultaneous attack (at some time the next day). Neither general will decide to attack unless he is sure that the other will attack with him. The generals can only communicate by means of a messenger. Normally, it takes the messenger one hour to get from one encampment to the other. However, it is possible that he will get lost in the dark or, worse yet, be captured by the enemy. Fortunately, on this particular night, everything goes smoothly. How long will it take them to coordinate an attack?

We now show that despite the fact that everything goes smoothly, no agreement can be reached and no general can decide to attack. (This is, in a way, a folk theorem of operating systems theory; cf. [16, 17, 44].) Suppose General A sends a message to General B saying "Let's attack at dawn," and the messenger delivers it an hour later. General A does not immediately know whether the messenger succeeded in delivering the message. And because B would not attack at dawn if the messenger is captured and fails to deliver the message, A will not attack unless he knows that the message was successfully delivered. Consequently, B sends the messenger back to A with an acknowledgment. Suppose the messenger delivers the acknowledgment to A an hour later. Since B knows that A will not attack without knowing that B received the original message, he knows that A will not attack unless the acknowledgment is successfully delivered. Thus, B will not attack unless he knows that the acknowledgment has been successfully delivered. However, for B to know that the acknowledgment has been successfully delivered, A must send the messenger back with an acknowledgment to the acknowledgment.... Similar arguments can be used to show that no fixed finite number of acknowledgments, acknowledgments to acknowledgments, etc., suffices for the generals to attack. Note that in the discussion above, the generals are essentially running a *handshake* protocol (cf. [17]). The above discussion shows that for no k does a k-round handshake protocol guarantee that the generals be able to coordinate an attack.

In fact, we can use this intuition to actually prove that the generals can never attack and be guaranteed that they are attacking simultaneously. We argue by induction on d—the number of messages delivered by the time of the attack—that d messages do not suffice. Clearly, if no message is delivered, then B will not know of the intended attack, and a simultaneous attack is impossible. For the inductive step, assume that k messages do not suffice. If $k + 1$ messages suffice, then the sender of the $(k + 1)$st message attacks without knowing whether his last message arrived. Since whenever one general attacks they both do, the intended receiver of the $(k + 1)$st message must attack regardless of whether the $(k + 1)$st message is delivered. Thus, the $(k + 1)$st message is irrelevant, and k messages suffice, contradicting the inductive hypothesis.

After presenting a detailed proof of the fact that no protocol the generals can use will satisfy their requirements and allow them to coordinate an attack, Yemini and Cohen in [44] make the following remark:

> ... Furthermore, proving protocols correct (or impossible) is a difficult and cumbersome art in the absence of proper formal tools to reason about protocols. Such backward-induction argument as the one used in the impossibility proof should require less space and become more convincing with a proper set of tools.

Yemini and Cohen's proof does not explicitly use reasoning about knowledge, but it uses a many-scenarios argument to show that if the generals both attack in one scenario, then there is another scenario in which one general will attack and the other will not. The crucial point is that the actions that should be taken depend not only on the actual state of affairs (in this case, the messenger successfully delivering the messages), but also (and in an acute way) on what other states of affairs the generals consider *possible*. Knowledge is just the dual of possibility, so reasoning about knowledge precisely captures the many-scenario argument in an intuitive way. We feel that understanding the role knowledge plays in problems such as coordinated attack is a first step towards simplifying the task of designing and proving the correctness of protocols.

A protocol for the coordinated attack problem, if one did exist, would ensure that when the generals attack, they are guaranteed to be attacking simultaneously. Thus, in a sense, an attacking general (say A) would know that the other general (say B) is also attacking. Furthermore, A would know that B similarly knows that A is attacking. It is easy to extend this reasoning to show that when the generals attack they have common knowledge of the attack. However, each message that the messenger delivers can add at most one level of knowledge about the desired attack, and no more. For example, when the message is first delivered to B, B knows about A's desire to coordinate an attack, but A does not know whether the message was delivered, and therefore A does not know that B knows about the intended attack. And when the messenger returns to A with B's acknowledgment, A knows that B knows about the intended attack, but, not knowing whether the messenger delivered the acknowledgment, B does not know that A knows (that B knows of the intended attack). This in some sense explains why the generals cannot reach an agreement to attack using a finite number of messages. We are about to formalize this intuition. Indeed, we shall prove a more general result from which the inability to achieve a guaranteed coordinated attack follows as a corollary. Namely, we prove that communication cannot be used to attain common knowledge in a system in which communication is not guaranteed, and show that, in order to coordinate an attack, common knowledge is required. Before we do so, we need to define some of the terms that we use more precisely.

5. *A General Model of a Distributed System*

We now present a general model of a distributed environment. Formally, we model such an environment by a distributed system, where the agents are taken to be processors and interaction between agents is modeled by messages sent between the processors over communication links. For the sake of generality and applicability to problems involving synchronization in distributed systems, our treatment will allow processors to have hardware clocks. Readers not interested in such issues can safely ignore all reference to clocks made throughout the paper.

We view a distributed system as a finite collection $\{p_1, p_2, \ldots, p_n\}$ of two or more processors that are connected by a communication network. We assume an external source of "real time" that in general is not directly observable by the processors. The processors are state machines that possibly have *clocks*, where a clock is a monotone nondecreasing function of real time. If a processor has a clock, then we assume that its clock reading is part of its state. (This is in contrast to the approach taken by Neiger and Toueg in [38]; the difference is purely a matter of taste.) The processors communicate with each other by sending messages along the links in the network.

A *run* r of a distributed system is a description of an execution of the system, from time 0 until the end of the execution. (We assume for simplicity that the system executes forever. If it terminates after finite time, we can just assume that it remains in the same state from then on.) A *point* is a pair (r, t) consisting of a run r and a time $t \geq 0$. We characterize the run r by associating with each point (r, t) every processor p_i's *local history* at (r, t), denoted $h(p_i, r, t)$. Roughly speaking, $h(p_i, r, t)$ consists of the sequence of events that p_i has observed up to time t in run r. We now formalize this notion. We assume that processor p_i "wakes up" or joins the system in run r at some time $t_{\text{init}}(p_i, r) \geq 0$. The processor's local state when it wakes up is called its *initial state*. The *initial configuration* of a run consists of the initial state and the wake up time for each processor. In systems with clocks, the *clock time function* τ describes processors' clock readings; $\tau(p_i, r, t)$ is the reading of p_i's clock at the point (r, t). Thus, $\tau(p_i, r, t)$ is undefined for $t < t_{\text{init}}(p_i, r)$ and is a monotonic nondecreasing function of t for $t \geq t_{\text{init}}(p_i, r)$. We say that r and r' have the same clock readings if $\tau(p_i, r, t) = \tau(p_i, r', t)$ for all processors p_i and all times t. (If there are no clocks in the system, we say for simplicity that all runs have the same clock readings.) We take $h(p_i, r, t)$ to be empty if $t < t_{\text{init}}(p_i, r)$. For $t \geq t_{\text{init}}(p_i, r)$, the history $h(p_i, r, t)$ consists of p_i's initial state and the sequence of messages p_i has sent and received up to, but not including, those sent or received at time t (in the order they were sent/received). We assume that this sequence of messages is finite. If p_i has a clock, the messages are also marked with the time at which they were sent or received (i.e., with $\tau(p_i, r, t)$, if they were sent or received at time t), and the history includes the range of values that the clock has read up to and including time t. If we consider randomized protocols, then $h(p_i, r, t)$ also includes p_is, random coin tosses. For ease of exposition, we restrict attention to deterministic protocols in this paper. In a deterministic system with no external inputs and no failures, a processor's initial state will be a function of its history. Thus, the sequence of internal states that a processor goes through can be recovered from its history.

Corresponding to every distributed system, given an appropriate set of assumptions about the properties of the system and its possible interaction with its environment, there is a natural set R of all possible runs of the system. We identify a distributed system with such a set R of its possible runs. For ease of exposition, we sometimes slightly abuse the language and talk about a point (r, t) as being *a point of* R when $r \in R$. A run r' is said to *extend* a point (r, t) if $h(p_i, r, t') = h(p_i, r', t')$ for all $t' \leq t$ and all processors p_i. Observe that r' extends (r, t) iff r extends (r', t).

Identifying a system with a set of runs is an important idea that will play a crucial role in allowing us to make precise the meaning of knowledge in a distributed system. The relative behavior of clocks, the properties of communication in the system, and many other properties of the system, are directly reflected in the properties of this set of runs. Thus, for example, a system is synchronous exactly if in all possible runs of the system the processors and the communication medium work in synchronous phases. A truly asynchronous system is one in which the set of runs allows any message sent to be delayed an unbounded amount of time before being delivered. (We discuss asynchrony in greater detail in Section 8.) Clocks are guaranteed to be synchronized to within a bound of δ if they differ by no more than δ time units at all points in all runs of the system. If we view the set of runs as a probability space with some appropriate measure, then we can also capture probabilistic properties of the environment and formalize probabilistic protocols in this framework.

We shall often be interested in the set of runs generated by running a particular *protocol*, under some assumptions on the communication medium. Intuitively, a protocol is a function specifying what actions a processor takes (which in our case amounts to what messages it sends) at any given point (after the processor wakes up) as a function of the processor's local state. Since a processor's local state is determined by its history, we simply define a protocol to be a deterministic function specifying what messages the processor should send at any given instant, as a function of the processor's history. Recall that $h(p_i, r, t)$, processor p_i's history at the point (r, t), does not include messages sent or received at time t, so a processor's actions at time t according to a protocol depend only on messages received in the past. As we mentioned above, for ease of exposition we restrict attention to deterministic protocols in this paper. The definitions and results can be extended to nondeterministic and probabilistic protocols in a straightforward way. A *joint protocol for G* is a tuple consisting of a protocol for every processor in G.

6. *Ascribing Knowledge to Processors*

What does it mean to say that a processor *knows* a fact φ? In our opinion, there is no unique "correct" answer to this question. Different interpretations of knowledge in a distributed system are appropriate for different applications. For example, an interpretation by which a processor is said to know φ only if φ appears explicitly in a designated part of the processor's storage (its "database") seems interesting for certain applications. In other contexts, we may be interested in saying that a processor knows φ if the processor could deduce φ from the information available to it. In this section, we give precise definitions of interpretations of knowledge in a distributed system.

We assume the existence of an underlying logical language of formulas for representing *ground facts* about the system. A ground fact is a fact about the state of the system that does not explicitly involve processors' knowledge. For example, "*the value of register x is* 0" or "*processor p_i sent the message m to processor p_j*" are ground facts.

We extend the original language of ground formulas to a language that is closed under operators for knowledge, distributed knowledge, everyone knows, and common knowledge (so that for every formula φ, processor p_i, and subset G of the processors $K_i\varphi$, $D_G\varphi$, $E_G\varphi$, and $C_G\varphi$ are formulas), and under Boolean connectives. (In Section 11, we consider additional operators.)

We now describe one of the most natural ways of ascribing knowledge to processors in a distributed system, which we call *view-based* knowledge interpretations. At every point each processor is assigned a *view*; we say that two points are *indistinguishable* to the processor if it has the same view in both. A processor is then said to *know* a fact at a given point exactly if the fact holds at all of the points that the processor cannot distinguish from the given one. Roughly speaking, a processor knows all of the facts that (information theoretically) follow from its view at the current point.[2]

[2] In a previous version of this paper [22], view-based knowledge interpretations were called *state-based* interpretations. Particular view-based knowledge interpretations were first suggested to us independently by Cynthia Dwork and by Stan Rosenschein. Since the appearance of [22], most authors who considered knowledge in distributed systems have focused on view-based interpretations; cf. [4], [9], [14], [19], [20], [28], [36], [40], and [42] for an overview. (See [11] and [34] for examples of interpretations of knowledge that are not view based.) The approach taken to defining knowledge in view-based systems is closely related to the possible-worlds approach taken by Hintikka [25]. For us, the "possible worlds" are the points in the system; the "agents" are the processors. A processor in one world (i.e., point) considers another world possible if it has the same view in both.

More formally, a *view function* v for a system R assigns to every processor at any given point of R a view from some set Σ of *views* (the structure of Σ is not relevant at this point); that is, $v(p_i, r, t) \in \Sigma$ for each processor p_i and point (r, t) of R. Given that a processor's history captures all of the events in the system that a processor may possibly observe, we require that the processor's view at any given point be a function of its history at that point. In other words, whenever $h(p_i, r, t) = h(p_i, r', t')$, it must also be the case that $v(p_i, r, t) = v(p_i, r', t')$.

A *view-based knowledge interpretation* $\mathcal{I}$ is a triple (R, π, v), consisting of a set of runs R, an assignment π that associates with every point in R a truth assignment to the ground facts (so that for every point (r, t) in R and every ground fact P, we have $\pi(r, t)(P) \in \{\textbf{true}, \textbf{false}\}$), and a view function v for R. A triple $(\mathcal{I}, r, t)$, where $\mathcal{I}$ is a knowledge interpretation and (r, t) is a point of R, is called a *knowledge point*. Formulas are said to be true or false of knowledge points. Let $\mathcal{I} = (R, \pi, v)$. We can now define the truth of a formula φ at a knowledge point $(\mathcal{I}, r, t)$, denoted $(\mathcal{I}, r, t) \models \varphi$ (and also occasionally read "φ *holds at* $(\mathcal{I}, r, t)$", or just "φ *holds at* (r, t)," if the interpretation $\mathcal{I}$ is clear from context), by induction on the structure of formulas:

(a) If P is a ground formula then $(\mathcal{I}, r, t) \models P$ iff $\pi(r, t)(P) = \textbf{true}$.
(b) $(\mathcal{I}, r, t) \models \neg\psi$ iff $(\mathcal{I}, r, t) \not\models \psi$.
(c) $(\mathcal{I}, r, t) \models \psi_1 \wedge \psi_1$ iff $(\mathcal{I}, r, t) \models \psi_1$ and $(\mathcal{I}, r, t) \models \psi_2$.
(d) $(\mathcal{I}, r, t) \models K_i\psi$ iff $(\mathcal{I}, r', t') \models \psi$ for all (r', t') in R satisfying $v(p_i, r, t) = v(p_i, r', t')$.

Part (a) says the truth value of ground facts is defined by π. Parts (b) and (c) state that negations and conjunctions have their classical meaning. Part (d) captures the fact a processor p_i's knowledge at a point (r, t) is completely determined by its view $v(p_i, r, t)$. The processor does not know φ in a given view exactly if there is a point (in R) at which the processor has that same view, and φ does not hold. The definitions of when $E_G\psi$ and $C_G\psi$ hold at a knowledge point follow directly from the definition of E_G and C_G in Section 3:

(e) $(\mathcal{I}, r, t) \models E_G\psi$ iff $(\mathcal{I}, r, t) \models K_i\psi$ for all $p_i \in G$.
(f) $(\mathcal{I}, r, t) \models C_G\psi$ iff $(\mathcal{I}, r, t) \models E_G^k\psi$ for all $k > 0$.

Let us consider when a group G of processors has distributed knowledge of a fact. Intuitively, a group's distributed knowledge is the combined knowledge of all of its members. For example, we could imagine considering the group as being able to distinguish two points if one (or more) of its members can distinguish them. The set of points indistinguishable by G from the current one is then the intersection of the sets of points indistinguishable by the individual members of the group. We can therefore define when a group G has distributed knowledge of a fact ψ as follows:

(g) $(\mathcal{I}, r, t) \models D_G\psi$ iff $(\mathcal{I}, r', t') \models \psi$ for all (r', t') in R satisfying $v(p_i, r, t) = v(p_i, r', t')$ for all $p_i \in G$.

Notice that indeed under this definition, if one member of G knows φ while another member knows that $\varphi \supset \psi$, then the members of G have distributed knowledge of ψ. The definition of distributed knowledge given above is in a precise sense a direct generalization of individual processors' knowledge in clause (d) above. We can define the *joint view assigned by* v *to* G to be

$$v(G, r, t) \stackrel{\text{def}}{=} \{\langle p_i, v(p_i, r, t)\rangle : p_i \in G\}.$$

It is easy to check that $(\mathscr{I}, r, t) \models D_G\psi$ iff $(\mathscr{I}, r', t') \models \psi$ for all (r', t') in R satisfying $v(G, r, t) = v(G, r', t')$. Thus, we can identify the distributed knowledge of a group G with the knowledge of an agent whose view is the group's joint view.[3] Note that the knowledge distributed in a group of size one coincides with its unique member's knowledge.

View-based interpretations will prove to be a useful way of ascribing knowledge to processors for the purpose of the design and analysis of distributed protocols. We now discuss some of the basic properties of knowledge in view-based interpretations. Fix a system R and a view function v. We can construct a graph corresponding to R and v by taking the nodes of the graph to be all the points of R, and joining two nodes (r, t) and (r', t') by an edge labeled p_i if $v(p_i, r, t) = v(p_i, r', t')$; that is, if p_i has the same view at both points. Our definition of knowledge under a view-based interpretation immediately implies that $K_i\varphi$ holds at a given point (r, t) if and only if φ holds at all points (r', t') that share an edge labeled p_i with (r, t). Define a point (r', t') in this graph to be *G-reachable from* (r, t) *in k steps* (*with respect to the view function* v) if there exist points (r_0, t_0), $(r_1, t_1), \ldots, (r_k, t_k)$ such that $(r, t) = (r_0, t_0)$, $(r', t') = (r_k, t_k)$, and for every $i < k$ there is a processor $p_{j_i} \in G$ such that (r_i, t_i) and (r_{i+1}, t_{i+1}) are joined by an edge labeled p_{j_i}. It follows that $E_G\varphi$ holds at (r, t) under this view-based interpretation exactly if φ holds at all points G-reachable from (r, t) in 1 step. An easy induction on k shows that $E_G^k\varphi$ holds exactly if φ holds at all points G-reachable in k steps. Consequently, it is easy to see that $C_G\varphi$ holds at a point (r, t) if and only if φ holds at all points that are G-reachable from (r, t) in a finite number of steps. In the particular case that G is the set of all processors, then $C_G\varphi$ holds at (r, t) exactly if φ holds at all points in the same connected component of the graph as (r, t).

The way distributed knowledge is represented in this graph is also instructive: $D_G\varphi$ holds at a given point (r, t) iff φ holds at all points (r', t') such that for each $p_i \in G$, there is an edge between (r, t) and (r', t') labeled by p_i. Thus, for distributed knowledge the set of points we need to consider is the intersection of the sets of points we consider when determining what facts each individual processor knows.

By describing the various notions of knowledge in the view-based case via this graph, it becomes easier to investigate their properties. In fact, this graph is very closely related to *Kripke structures*, a well-known standard way of modeling modal logics. In fact, drawing on the theory of modal logics, we can immediately see that the definition of knowledge in view-based interpretations agrees with the well-known modal logic S5 (cf. [23]). A modal operator M is said to *have the properties of S5* if it satisfies the following axioms and rules of inference:

A1. The *knowledge axiom*: $M\varphi \supset \varphi$,
A2. The *consequence closure axiom*: $M\varphi \wedge M(\varphi \supset \psi) \supset M\psi$,
A3. The *positive introspection axiom*: $M\varphi \supset MM\varphi$,
A4. The *negative introspection axiom*: $\neg M\varphi \supset M\neg M\varphi$, and
R1. The *rule of necessitation*: From φ infer $M\varphi$.

Given a knowledge interpretation $\mathscr{I}$ for a system R, a fact ψ is said to be *valid in the system* if it holds at $(\mathscr{I}, r, t)$ for all points (r, t) of R. In our context the rule R1 means that whenever φ is valid in the system, so is $M\varphi$.

We can now show:

PROPOSITION 1. *Under view-based knowledge interpretations, the operators* K_i, D_G, *and* C_G *all have the properties of S5.*

[3] The knowledge ascribed to a set of processes by Chandy and Misra in [4] essentially corresponds to the distributed knowledge of that set, as defined here. See also [40] and [42].

The proof is a consequence of the fact that the definitions of these notions are based on equivalence relations (over points): The relation of processor p_i's having the same view at two points, the relation of all processors in G having the same joint views at both points, and the relation of being reachable via a path consisting solely of edges labeled by members of G in the graph corresponding to the view, are all equivalence relations. The proof of this proposition can be found in [23].

In addition to having the properties of S5, common knowledge has two additional useful properties under view-based interpretations:

C1. The *fixed point axiom*: $C_G\varphi \equiv E_G(\varphi \wedge C_G\varphi)$, and
C2. The *induction rule*: From $\varphi \supset E_G(\varphi \wedge \psi)$ infer $\varphi \supset C_G\psi$.

The fixed-point axiom essentially characterizes $C_G\varphi$ as the solution of a fixed-point equation (in fact, it is the greatest solution; we discuss this in more detail in Section 11 and Appendix A). This property of common knowledge is crucial in many of our proofs.

Intuitively, the induction rule says that if φ is "public" and implies ψ, so that whenever φ holds then everybody knows $\varphi \wedge \psi$, then whenever φ holds, ψ is common knowledge. We call it the "induction rule" because it is closely related to the notion of induction in arithmetic: Using the fact that $\varphi \supset E_G(\varphi \wedge \psi)$ is valid in the system, we can prove by induction on k that $\varphi \supset E_G^k(\varphi \wedge \psi)$ is also valid in the system, for all $k > 0$. It then follows that $\varphi \supset C_G\psi$ is valid in the system. Roughly speaking, this proof traces our line of reasoning when we argued that the children in the muddy children puzzle attain common knowledge of the father's statement. We can get an important special case of the Induction Rule by taking ψ to be φ. Since $E_G(\varphi \wedge \varphi)$ is equivalent to $E_G\varphi$, we get that from $\varphi \supset E_G\varphi$ we can infer $\varphi \supset C_G\varphi$.

A very important instance of view-based knowledge interpretations, that will be used extensively from Section 11 on, is called the *complete-history interpretation*. Under this interpretation we have $v(p_i, r, t) \stackrel{\text{def}}{=} h(p_i, r, t)$. That is, the processor's complete history is taken to be the view on which the processor's knowledge is based. (In a previous version of this paper [22], this was called the *total view* interpretation.) The complete-history interpretation makes the finest possible distinctions among histories. Thus, in a precise sense, it provides the processors with at least as much knowledge about the ground formulas as any other view-based interpretation. This is one of the reasons why the complete-history interpretation is particularly well suited for proving possibility and impossibility of achieving certain goals in distributed systems, and for the design and analysis of distributed protocols (cf. [4], [9], and [36]).

Notice that view-based knowledge interpretations ascribe knowledge to a processor without the processor necessarily being "aware" of this knowledge, and without the processor needing to perform any particular computation in order to obtain such knowledge. Interestingly, even if the view function v does not distinguish between possibilities at all, that is, if there is a single view Λ such that $v(p_i, r, t) = \Lambda$ for all p_i, r, and t, the processors are still ascribed quite a bit of knowledge: Every fact that is true at all points of the system is common knowledge among all the processors under this view-based interpretation (and in fact under all view-based interpretations). Note that the hierarchy of Section 3 collapses under this interpretation, with $D\varphi \equiv E\varphi \equiv C\varphi$. This interpretation makes the coarsest possible distinctions among histories; at the other extreme we have the complete-history interpretation, which makes the finest possible distinctions among histories.

Another reasonable view-based interpretation is one in which $v(p_i, r, t)$ is defined to be p_i's local state at (r, t). (Recall that processors are state machines, and are thus assumed to be in some local state at every point.) This is the choice made in [14], [41], and [42]. Under this interpretation, a processor might "forget" facts that it knows. In particular, if a processor can arrive at a given state by two different message histories, then, once in that state, the processor's knowledge cannot distinguish between these two "possible pasts." In the complete-history interpretation, a processor's view encodes all of the processor's previous states, and therefore processors do not forget what they know; if a processor knows φ at a knowledge point $(\mathcal{I}, r, t)$, then at all knowledge points $(\mathcal{I}, r, t')$ with $t' > t$ the processor will know that it once knew φ. Thus, while there may be temporary facts such as "it is 3 on my clock" which a processor will not know at 4 o'clock, it will know at 4 o'clock that it previously knew that it was 3 o'clock.

Other view-based interpretations that may be of interest are ones in which a processor's view is identified with the contents of its memory, or with the position of its program counter (see [26] for a closer look at some of these view-based interpretations). The precise view-based interpretations we choose will vary from application to application. For proving lower bounds we frequently use the complete-history interpretation since, in general, if processors cannot perform an action with the knowledge they have in the complete-history interpretation, they cannot perform it at all. On the other hand, if we can show that very little information is required to perform a given action, this may suggest an efficient protocol for performing it.

Although view-based knowledge interpretations are natural and useful in many applications, they do not cover all reasonable possibilities of ascribing knowledge to processors in a distributed system. For example, as we have commented above, view-based knowledge interpretations ascribe knowledge to processors in a fashion that is independent of the processor's computational power. To the extent that we intend processors' knowledge to correspond closely to the actions they can perform, it often becomes crucial to define knowledge in a way that depends on the processors' computational powers (cf. [34] and [36]). In most of the paper, we deal exclusively with view-based knowledge interpretations. However, in order to be able to prove stronger negative results about the attainability of certain states of knowledge, we now give a general definition of knowledge interpretations, which we believe covers all reasonable cases.

Intuitively, we want to allow any interpretation that satisfies the two properties discussed in Section 3: (1) that a processor's knowledge be a function of its history and (2) that only true things be known (so that the axiom $K_i\varphi \supset \varphi$ is valid). We capture the first property through the notion of an epistemic interpretation. An *epistemic interpretation* $\mathcal{I}$ is a function assigning to every processor p_i at any given point (r, t), a set $\mathcal{K}_i^{\mathcal{I}}(r, t)$ of facts in the extended language that p_i is said to "believe." $\mathcal{K}_i^{\mathcal{I}}(r, t)$ is required to be a function of p_i's history at (r, t). Thus, if $h(p_i, r, t) = h(p_i, r', t')$, then $\mathcal{K}_i^{\mathcal{I}}(r, t) = \mathcal{K}_i^{\mathcal{I}}(r', t')$.

Given an epistemic interpretation $\mathcal{I}$, we now specify when a formula φ of the extended language holds at a point (r, t) (denoted $(\mathcal{I}, r, t) \models \varphi$). As before, if φ is a ground fact, we say that $(\mathcal{I}, r, t) \models \varphi$ iff $\pi(r, t)(\varphi) = \textbf{true}$, while if φ is a conjunction or a negation, then its truth is defined based on the truth of its subformulas in the obvious way. If φ is of the form $K_i\psi$, then $(\mathcal{I}, r, t) \models K_i\psi$ iff $\psi \in \mathcal{K}_i^{\mathcal{I}}(r, t)$. In this case we say that p_i *believes* ψ. The formula $E_G\psi$ is identified with the conjunction $\bigwedge_{p_i \in G} K_i\psi$, so that $(\mathcal{I}, r, t) \models E_G\psi$ iff $(\mathcal{I}, r, t) \models K_i\psi$ for all

$p_i \in G$. If φ is of the form $C_G\psi$, then $(\mathcal{I}, r, t) \models C_G\psi$ iff $(\mathcal{I}, r, t) \models E_G(\psi \wedge C_G\psi)$. Thus, common knowledge is defined so that the fixed point axiom holds, rather than as an infinite conjunction. Although this definition seems circular, it is not. In order to determine if $(\mathcal{I}, r, t) \models C_G\psi$, we first check if $(\mathcal{I}, r, t) \models K_i(\psi \wedge C_G\psi)$ for all $p_i \in G$. The latter fact can be determined by considering the sets $\mathscr{K}_i^{\mathcal{I}}(r, t)$. Finally, to handle distributed knowledge, we need to add a set $\mathscr{K}_G^{\mathcal{I}}(r, t)$ of formulas to every point (r, t) for each set of processors G, analogous to the sets $\mathscr{K}_i^{\mathcal{I}}(r, t)$ for individual processors. We define $(\mathcal{I}, r, t) \models D_G\varphi$ if $\varphi \in \mathscr{K}_G^{\mathcal{I}}(r, t)$. The sets $\mathscr{K}_G^{\mathcal{I}}(r, t)$ must be a function of G's joint history at (r, t). We may want to put some restrictions on the sets $\mathscr{K}_G^{\mathcal{I}}(r, t)$. For example, we may require that, if $i \in G$ and $\varphi \in \mathscr{K}_i^{\mathcal{I}}(r, t)$, then $\varphi \in \mathscr{K}_G^{\mathcal{I}}(r, t)$ (which implies that $K_i\varphi \supset D_G\varphi$ is valid). Since we do not consider distributed knowledge in interpretations that are not view based, we do not pursue the matter any further here.

The knowledge axiom $K_i\varphi \supset \varphi$ is not necessarily valid in epistemic interpretations. Indeed, that is why we have interpreted $K_i\varphi$ as "processor i believes φ" in epistemic interpretations, since the knowledge axiom is the key property that is taken to distinguish knowledge from belief. A processor's beliefs may be false, although a processor cannot be said to *know* φ if φ is in fact false. Given an epistemic interpretation $\mathcal{I}$ and a set of runs R, we say that $\mathcal{I}$ is a *knowledge interpretation for* R if for all processors p_i, times t, runs $r \in R$ and formulas φ in the extended language, it is the case that whenever $(\mathcal{I}, r, t) \models K_i\varphi$ holds, $(\mathcal{I}, r, t) \models \varphi$ also holds. Thus, an epistemic interpretation for R is a knowledge interpretation for R exactly if it makes the knowledge axiom valid in R. Notice that the view-based knowledge interpretations defined above are in particular knowledge interpretations.

A trivial consequence of our definition above is:

LEMMA 2. *Let $\mathcal{I}$ be a knowledge interpretation for R and let (r, t) be a point of R. The following are equivalent for a nonempty subset G of processors:*

(1) $(\mathcal{I}, r, t) \models C_G\varphi$
(2) $(\mathcal{I}, r, t) \models K_i(\varphi \wedge C_G\varphi)$ *for all processors* $p_i \in G$
(3) $(\mathcal{I}, r, t) \models K_i(\varphi \wedge C_G\varphi)$ *for some processor* $p_i \in G$.

This lemma shows that common knowledge requires simultaneity in a very strong sense: When a new fact becomes common knowledge in a group G, the local histories of all of the members of G must change *simultaneously* to reflect the event of the fact's becoming common knowledge. This point is perhaps best understood if we think of time as ranging over the natural numbers. Given a knowledge interpretation $\mathcal{I}$, suppose that common knowledge does not hold at the point (r, t) but does hold at the point $(r, t + 1)$, so that $(\mathcal{I}, r, t) \models \neg C_G\varphi$ and $(\mathcal{I}, r, t + 1) \models C_G\varphi$. Then it must be the case that the local histories of all processors in G changed between times t and $t + 1$. To see this, note that by Lemma 2 we have $(\mathcal{I}, r, t + 1) \models K_i(\varphi \wedge C_G\varphi)$ for all $p_i \in G$. Suppose $p_i \in G$ has the same local history in (r, t) and $(r, t + 1)$. Then by our assumption that a processor's knowledge depends only on its local history, we have that $(\mathcal{I}, r, t) \models K_i(\varphi \wedge C_G\varphi)$. Now by Lemma 2 again, we have $(\mathcal{I}, r, t) \models C_G\varphi$, contradicting our original assumption.

We close this section with another trivial observation that follows easily from Lemma 2.

LEMMA 3. *Let $\mathcal{I}$ be a knowledge interpretation for R, let r and r' be runs in R, and let p_i be a processor in G. If $h(p_i, r, t) = h(p_i, r', t')$, then $(\mathcal{I}, r, t) \models C_G\varphi$ iff $(\mathcal{I}, r', t') \models C_G\varphi$.*

PROOF. Given that $p_i \in G$, we have by Lemma 2 that $(\mathcal{I}, r, t) \models C_G\varphi$ iff $(\mathcal{I}, r, t) \models K_i(\varphi \wedge C_G\varphi)$. Since $h(p_i, r, t) = h(p_i, r', t')$, this holds iff $(\mathcal{I}, r', t') \models K_i(\varphi \wedge C_G\varphi)$. Again by Lemma 2 this is true iff $(\mathcal{I}, r', t') \models C_G\varphi$, and we are done. □

7. *Coordinated Attack Revisited*

Now that we have the basic terminology with which to define distributed systems and knowledge in distributed systems, we can relate the ability to perform a coordinated attack to the attainment of common knowledge of particular facts. This in turn will motivate an investigation of the attainability of common knowledge in systems of various types.

We formalize the coordinated attack problem as follows: We consider the generals as processors and their messengers as communication links between them. The generals are assumed to each behave according to some predetermined deterministic protocol; that is, a general's actions (what messages it sends and whether it attacks) at a given point are a deterministic function of his history and the time on his clock. In particular, we assume that the generals are following a joint protocol (P_A, P_B), where A follows P_A and B follows P_B. We can thus identify the generals with a distributed system R, consisting of all possible runs of (P_A, P_B). According to the description of the coordinated attack problem in Section 4, the divisions do not initially have plans to attack. Formally, this means that the joint protocol the generals are following has the property that in the absence of any successful communication neither general will attack. Thus, in any run of R where no messages are delivered, the generals do not attack.

We can now show that attacking requires attaining common knowledge of the attack:

PROPOSITION 4. *Any correct protocol for the coordinated attack problem has the property that whenever the generals attack, it is common knowledge that they are attacking.*

PROOF. Let (P_A, P_B) be a correct (joint) protocol for the coordinated attack problem, with R being the corresponding system. Consider a ground language consisting of a single fact $\psi \stackrel{\text{def}}{=}$ "*both generals are attacking*," let $\pi(r, t)$ assign a truth value to this formula in the obvious way at each point (r, t), and let $\mathcal{I}$ be the corresponding complete-history interpretation. Assume that the generals attack at the point $(\hat{r}, \hat{t})$ of R. We show that $(\mathcal{I}, \hat{r}, \hat{t}) \models C\psi$. Our first step is to show that $\psi \supset E\psi$ is valid in the system R. Assume that (r, t) is an arbitrary point of R. If $(\mathcal{I}, r, t) \models \neg\psi$, then we trivially have $(\mathcal{I}, r, t) \models \psi \supset E\psi$. If $(\mathcal{I}, r, t) \models \psi$, then both generals attack at (r, t). Suppose that (r', t') is a point of R in which A has the same local history as in (r, t). Since A is executing a deterministic protocol and A attacks in (r, t), A must also attack in (r', t'). Furthermore, given that the protocol is a correct protocol for coordinated attack, if A attacks in (r', t'), then so does B, and hence $(\mathcal{I}, r', t') \models \psi$. It follows that $(\mathcal{I}, r, t) \models K_A\psi$; similarly we obtain $(\mathcal{I}, r, t) \models K_B\psi$. Thus $(\mathcal{I}, r, t) \models E\psi$, and again we have $(\mathcal{I}, r, t) \models \psi \supset E\psi$. We have now shown that $\psi \supset E\psi$ is valid in R. By the induction rule it follows that $\psi \supset C\psi$ is also valid in R. Since $(\mathcal{I}, \hat{r}, \hat{t}) \models \psi$, we have that $(\mathcal{I}, \hat{r}, \hat{t}) \models C\psi$ and we are done. □

Proposition 4 shows that common knowledge is a prerequisite for coordinated attack. Unfortunately, common knowledge is not always attainable, as we show in

the next section. Indeed, it is the unattainability of common knowledge that is the fundamental reason why the generals cannot coordinate an attack.

8. *Attaining Common Knowledge*

Following the coordinated attack example, we first consider systems in which communcation is not guaranteed. Intuitively, communication is not guaranteed in a system if messages might fail to be delivered in an arbitrary fashion, independent of any other event in the system. Completely formalizing this intuition seems to be rather cumbersome (cf. [20]), and we do not attempt to do so here. For our purposes, a weak condition, which must be satisfied by any reasonable definition of the notion of communication not being guaranteed, will suffice. Roughly speaking, we take communication not being guaranteed to correspond to two conditions. The first says that it is always possible that from some point on no messages will be received. The second says that if processor p_i does not get any information to the contrary (by receiving some message), then p_i considers it possible that none of its messages were received.

Formally, given a system R, we say that *communication in R is not guaranteed* if the following two conditions hold:

NG1. For all runs r and times t, there exists a run r' extending (r, t) such that r and r' have the same initial configuration and the same clock readings, and no messages are received in r' at or after time t.

NG2. If in run r processor p_i does not receive any messages in the interval (t', t), then there is a run r' extending (r, t') such that r and r' have the same initial configuration and the same clock readings, $h(p_i, r, t'') = h(p_i, r', t'')$ for all $t'' \leq t$, and no processor $p_j \neq p_i$ receives a message in r' in the interval $[t', t)$.

Note that the requirement that r and r' have the same initial configuration already follows from the fact that r' extends (r, t) if all the processors have woken up by time t in run r. In particular, if we restricted attention to systems where all processors were up at time 0, we would not require this condition.

We can now show that in a system in which communication is not guaranteed, common knowledge is not attainable.

THEOREM 5. *Let R be a system in which communication is not guaranteed, let $\mathcal{I}$ be a knowledge interpretation for R, and let $|G| \geq 2$. Let r be a run of R, and let r^- be a run of R with the same initial configuration and the same clock readings as r, such that no messages are received in r^- up to time t. Then for all formulas φ, it is the case that $(\mathcal{I}, r, t) \models C_G\varphi$ iff $(\mathcal{I}, r^-, t) \models C_G\varphi$.*

PROOF. Fix φ. Without loss of generality, we can assume $p_1, p_2 \in G$. Let $d(r)$ be the number of messages received in r up to (but not including) time t. We show by induction on k that if $d(r) = k$, then $(\mathcal{I}, r, t) \models C_G\varphi$ iff $(\mathcal{I}, r^-, t) \models C_G\varphi$. We assume that all the runs mentioned in the remainder of the proof have the same initial configuration and the same clock readings as r. First, assume that $d(r) = 0$. Thus, no messages are received in r up to time t. Since r and r^- have the same initial configuration and clock readings, it follows that $h(p_1, r, t) = h(p_1, r^-, t)$. By Lemma 3, we have $(\mathcal{I}, r^-, t) \models C_G\varphi$ iff $(\mathcal{I}, r, t) \models C_G\varphi$, as desired.

Assume inductively that the claim holds for all runs $r' \in R$ with $d(r') = k$, and assume that $d(r) = k + 1$. Let $t' < t$ be the latest time at which a message is received in r before time t. Let p_j be a processor that receives a message at time t'

in r. Let p_i be a processor in G such that $p_i \neq p_j$ (such a p_i exists since $|G| \geq 2$). From property NG2 in the definition of communication not being guaranteed, it follows that there is a run $r' \in R$ extending (r, t') such that $h(p_i, r, t'') = h(p_i, r', t'')$ for all $t'' < t$ and all processors $p_k \neq p_i$ receive no messages in r' in the interval $[t', t)$. By construction, $d(r') \leq k$, so by the inductive hypothesis we have that $(\mathcal{I}, r^-, t) \models C_G\varphi$ iff $(\mathcal{I}, r', t) \models C_G\varphi$. Since $h(p_i, r, t) = h(p_i, r', t)$, by Lemma 3 we have that $(\mathcal{I}, r', t) \models C_G\varphi$ iff $(\mathcal{I}, r, t) \models C_G\varphi$. Thus, $(\mathcal{I}, r^-, t) \models C_G\varphi$ iff $(\mathcal{I}, r, t) \models C_G\varphi$. This completes the proof of the inductive step. □

Note that Theorem 5 does not say that no fact can become common knowledge in a system where communication is not guaranteed. In a system where communication is not guaranteed but there is a global clock to which all processors have access, then at 5 o'clock it becomes common knowledge that it is 5 o'clock.[4] However, the theorem does say that nothing can become common knowledge unless it is also common knowledge in the absence of communication. This is a basic property of systems with unreliable communication, and it allows us to prove the impossibility of coordinated attack.

COROLLARY 6. *Any correct protocol for the coordinated attack problem guarantees that neither party ever attacks* (!).

PROOF. Recall that communication between the generals is not guaranteed (i.e., it satisfies conditions NG1 and NG2 above), and we assume that in the absence of any successful communication neither general will attack. Thus, if we take ψ to be "*both generals are attacking*," then $C\psi$ does not hold at any point in a run in which no messages are received (since ψ does not hold at any point of that run). Theorem 5 implies that the generals will never attain common knowledge of ψ in any run, and hence by Proposition 4 the generals will never attack. □

It is often suggested that for any action for which $C\varphi$ suffices, there is a k such that $E^k\varphi$ suffices, as is the case in the muddy children puzzle. The coordinated attack problem shows that this is false. The generals can attain $E^k\varphi$ of many facts φ for an arbitrarily large k (for example, if the first k messages are delivered). However, simultaneous coordinated attack requires *common knowledge* (as is shown in Proposition 4); nothing less will do.

The requirement of simultaneous attack in the coordinated attack problem is a very strong one. It seems that real-life generals do not need a protocol that guarantees such a strong condition, and can probably make do with one that guarantees a nonsimultaneous attack. We may want to consider weakening this requirement in order to get something that is achievable. In Section 11, we use a variant of the argument used in Corollary 6 to show that no protocol can even guarantee that, if one party attacks, then the other will *eventually* attack! On the other hand, a protocol that guarantees that if one party attacks, then with high probability the other will attack *is* achievable, under appropriate probabilistic assumptions about message delivery. The details of such a protocol are straightforward and left to the reader.

[4] We remark that the possible presence of some sort of global clock is essentially all that stops us from saying that no fact can become common knowledge if it was not already common knowledge at the beginning of a run. See Proposition 13 in Appendix B and the discussion before it for conditions under which it is the case that no fact can become common knowledge that was not initially common knowledge.

We can prove a result similar to Theorem 5 even if communication is guaranteed, as long as there is no bound on message delivery times. A system R is said to be a system with *unbounded message delivery times* if condition NG2 of communication not guaranteed holds, and in addition we have:

NG1′. For all runs r and all times t, u, with $t \leq u$, there exists a run r' extending (r, t) such that r' has the same initial configuration and the same clock readings as r, and no messages are received in r' in the interval $[t, u]$.

Asynchronous systems are often defined to be systems with unbounded message delivery times (for example, in [15]). Intuitively, condition NG1′ says that it is always possible for no messages to be received for arbitrarily long periods of time, whereas condition NG1 says that it is always possible for no messages at all to be received from some time on. In some sense, we can view NG1 as the limit case of NG1′. Notice that both systems where communication is not guaranteed and systems with unbounded message delivery times satisfy condition NG2. The proof of Theorem 5 made use only of NG2, not NG1, so we immediately get:

THEOREM 7. *Let R be a system with unbounded message delivery times, let $\mathcal{I}$ be a knowledge interpretation for R, and let $|G| \geq 2$. Let r be a run of R, and let r^- be a run of R with the same initial configuration and the same clock readings as r, such that no messages are received in r^- up to time t. Then, for all formulas φ, it is the case that $(\mathcal{I}, r, t) \models C_G\varphi$ iff $(\mathcal{I}, r^-, t) \models C_G\varphi$.*

The previous results show that, in a strong sense, common knowledge is not attainable in a system in which communication is not guaranteed or, for that matter, in a system in which communication *is* guaranteed, but there is no bound on the message delivery times. However, even when all messages *are* guaranteed to be delivered within a fixed time bound, common knowledge can be elusive. To see this, consider a system consisting of two processors, R2 and D2, connected by a communication link. Moreover, (it is common knowledge that) communication is guaranteed. But there is some uncertainty in message delivery times. For simplicity, let us assume that any message sent from R2 to D2 reaches D2 either immediately or after exactly ϵ seconds; furthermore, assume that this fact is common knowledge. Now suppose that at time t_S, R2 sends D2 a message m that does not contain a timestamp, that is, does not mention t_S in any way. The message m is received by D2 at time t_D. Let $sent(m)$ be the fact "*the message m has been sent.*" D2 does not know $sent(m)$ initially. How does {R2, D2}'s state of knowledge of $sent(m)$ change with time?

At time t_D, D2 knows $sent(m)$. Because it might have taken ϵ time units for m to be delivered, R2 cannot be sure that D2 knows $sent(m)$ before $t_S + \epsilon$. Thus, $K_R K_D sent(m)$ holds at time $t_S + \epsilon$ *and no earlier.* D2 knows that R2 will not know that D2 knows $sent(m)$ before $t_S + \epsilon$. Because for all D2 knows m may have been delivered immediately (in which case $t_S = t_D$), D2 does not know that R2 knows that D2 knows $sent(m)$ before $t_D + \epsilon$. Since t_D might be equal to $t_S + \epsilon$, R2 must wait until $t_S + 2\epsilon$ before he knows that $t_D + \epsilon$ has passed. Thus, $K_R K_D K_R K_D sent(m)$ holds at time $t_S + 2\epsilon$ but no earlier. This line of reasoning can be continued indefinitely, and an easy proof by induction shows that before time $t_S + k\epsilon$, the formula $(K_R K_D)^k sent(m)$ does not hold, while at $t_S + k\epsilon$ it does hold. Thus, it "costs" ϵ time units to acquire every level of "R2 knows that D2 knows." Recall that $Csent(m)$ implies $(K_R K_D)^k sent(m)$ for every k. It follows that $Csent(m)$ will never be attained!

We can capture this situation using our formal model as follows. Let $MIN = \lfloor t_S/\epsilon \rfloor$, and consider the system with a countable set of runs $\{r_i, r'_i : i \text{ an integer with } i \geq -MIN\}$. If $i \geq -MIN$, then in run r_i, R2 sends the message m at time $t_S + i\epsilon$ and D2 receives it at the same time. In run r'_i, R2 again sends the message m at time $t_S + i\epsilon$, but D2 receives it at time $t_S + (i + 1)\epsilon$. (Note our choice of MIN guarantees that all messages are sent at time greater than or equal to 0.) If we assume that in fact the message in the example took ϵ time to arrive, then the run r'_0 describes the true situation. However, it is easy to see that at all times t, R2 cannot distinguish runs r_i and r'_i (in that its local state is the same at the corresponding points in the two runs, assuming that only message m is sent), while D2 cannot distinguish r_i and r'_{i-1} (provided $i - 1 \geq -MIN$).

Our discussion of knowledge in a distributed system is motivated by the fact that we can view processors' actions as being based on their knowledge. Consider an *eager* epistemic interpretation $\mathcal{I}$ under which R2 believes $Csent(m)$ as soon as it sends the message m, while D2 believes $Csent(m)$ as soon as it receives m. Clearly, $\mathcal{I}$ is not a knowledge interpretation, because it is not knowledge consistent (R2 might believe that D2 knows $sent(m)$, when in fact D2 does not). However, once D2 receives m, which happens at most ϵ time units after R2 starts believing $Csent(m)$, it is easy to see that $Csent(m)$ does indeed hold! In a sense, Lemma 2 says that attaining common knowledge requires a certain kind of "natural birth"; it is not possible to attain it consistently unless simultaneity is attainable. But if one is willing to give up knowledge consistency (i.e., abandon the $K_i\varphi \supset \varphi$ axiom) for short intervals of time, something very similar to common knowledge can be attained.

The existence of an interval of up to ϵ time units during which R2 and D2's "knowledge" might be inconsistent may have practical impact. If the processors need to act on the basis of whether $Csent(m)$ holds during the interval, they might not act in an appropriately coordinated way. This is a familiar problem in the context of distributed database systems. There, committing a transaction roughly corresponds to entering into an agreement that the transaction has taken place in the database. However, in general, different sites of the database commit transactions at different times (although usually all within a small time interval). When a new transaction is being committed there is a "window of vulnerability" during which different sites might reflect inconsistent histories of the database. However, once all sites commit the transaction, the history of the database that the sites reflect becomes consistent (at least as far as the particular transaction is concerned). In Section 13 we return to the question of when an "almost knowledge consistent" version of common knowledge can be safely used "as if it were" knowledge.

Returning to the R2–D2 example, note that it is the uncertainty in relative message delivery time that makes it impossible to attain common knowledge, and not the fact that communication is not instantaneous. If it were common knowledge that messages took *exactly* ϵ time units to arrive, then $sent(m)$ would be common knowledge at time $t_S + \epsilon$ (and the system would consist only of run r_1).

Another way of removing the uncertainty is by having a common (global) clock in the system. Suppose that there is such a clock. Consider what would happen if R2 sends D2 the following message m':

"This message is being sent at time t_S; m."

Since there is a global clock and it is guaranteed that every message sent by R2 is delivered within ϵ time units, the fact that R2 sent m' to D2 would again become

common knowledge at time $t_S + \epsilon$! In this case, the system would consist of two runs, r_0 and r_1. At time $t_S + \epsilon$, D2 would know which of the two was actually the case, although R2 would not (although D2 could tell him by sending a message).

It seems that common knowledge is attainable in the latter two cases due to the possibility of simultaneously making the transition from not having common knowledge to having common knowledge (at time $t_S + \epsilon$). The impossibility of doing so in the first case was the driving force behind the extra cost in time incurred in attaining each additional level of knowledge.

Lemma 2 already implies that when $C\varphi$ first holds all processors must come to believe $C\varphi$ simultaneously. In particular, this means that all of the processors' histories must change simultaneously. However, strictly speaking, practical systems cannot guarantee absolute simultaneity. In particular, we claim that essentially all practical distributed systems have some inherent temporal uncertainty. There is always some uncertainty about the precise instant at which each processor starts functioning, and about exactly how much time each message takes to be delivered. In Appendix B we give a precise formulation of the notion of *temporal imprecision*, which captures these properties, and use methods derived from [7] and [21] to prove the following result:

THEOREM 8. *Let R be a system with temporal imprecision, let $\mathcal{I}$ be a knowledge interpretation for R, and let $|G| \geq 2$. Then for all runs $r \in R$, times t, and formulas φ it is the case that $(\mathcal{I}, r, t) \models C_G\varphi$ iff $(\mathcal{I}, r, 0) \models C_G\varphi$.*

Since practical systems turn out to have temporal imprecision, Theorem 8 implies that, strictly speaking, common knowledge cannot be attained in practical distributed systems! In such systems, we have the following situation: A fact φ can be known to a processor without being common knowledge, or it can be common knowledge (in which case that processor also knows φ), but due to (possibly negligible) imperfections in the system's state of synchronization and its communication medium, there is no way of getting from the first situation to the second! Note that if there is a global clock, then there cannot be any temporal imprecision. Thus, it is consistent with Theorem 8 that common knowledge is attainable in a system with a global clock.

Observe that we can now show that, formally speaking, even people cannot attain common knowledge of any new fact! Consider the father publicly announcing **m** to the children in the muddy children puzzle. Even if we assume that it is common knowledge that the children all hear whatever the father says and understand it, there remains some uncertainty as to exactly when each child comes to know (or comprehend) the father's statement. Thus, it is easy to see that the children do not immediately have common knowledge of the father's announcement. Furthermore, for similar reasons, the father's statement can never become common knowledge.

9. *A Paradox?*

There is a close correspondence between agreements, coordinated actions, and common knowledge. We have argued that, in a precise sense, reaching agreements and coordinating actions in a distributed system requires attaining common knowledge of certain facts. However, in the previous section we showed that common knowledge *cannot be attained* in practical distributed systems! We are faced with a seemingly paradoxical situation on two accounts. First of all, these results are in contradiction with practical experience, in which operations such as

reaching agreement and coordinating actions are routinely performed in many actual distributed systems. It certainly seems as if these actions are performed in such systems without the designers having to worry about common knowledge (and despite the fact that we have proved that common knowledge is unattainable!). Secondly, these results seem to contradict our intuitive feeling that common knowledge *is* attained in many actual situations; for example, by the children in the muddy children puzzle.

Where is the catch? How can we explain this apparent discrepancy between our formal treatment and practical experience? What is the right way to interpret our negative results from the previous section? Is there indeed a paradox here? Or perhaps we are using a wrong or useless definition of common knowledge?

We believe that we do have a useful and meaningful definition of common knowledge. However, a closer inspection of the situation is needed in order to understand the subtle issues involved. First of all, we shall see that only rather strong notions of coordination in a distributed system require common knowledge. Common knowledge corresponds to absolutely simultaneous coordination, which is more than is necessary in many particular applications. For many other types of coordination, weaker states of knowledge suffice. In the coming sections, we investigate a variety of weaker states of knowledge that are appropriate for many applications. Furthermore, in many cases practical situations (and practical distributed systems) can be faithfully modeled by a simplified abstract model, in which common knowledge *is* attainable. In such a case, when facts become common knowledge in the abstract model, it may be perfectly safe and reasonable to consider them to be common knowledge when deciding on actions to be performed in the actual system. We discuss this in greater detail in Section 13.

10. *Common Knowledge Revisited*

In Section 8, we showed that common knowledge is not attainable in practical distributed systems under *any* reasonable interpretation of knowledge (i.e., in any epistemic interpretation). Our purpose in the coming sections is to investigate what states of knowledge *are* attainable in such systems. For that purpose, we restrict our attention to view-based interpretations of knowledge, since they seem to be the most appropriate for many applications in distributed systems. Under view-based interpretations, it seems useful to consider an alternative view of common knowledge.

Recall the children's state of knowledge of the fact $\mathbf{m}$ in the muddy children puzzle. If we assume that it is common knowledge that all children comprehend $\mathbf{m}$ simultaneously, then after the father announces $\mathbf{m}$, the children attain $C\mathbf{m}$. However, when they attain $C\mathbf{m}$ it is not the case that the children learn the infinitely many facts of the form $E^k\mathbf{m}$ separately. Rather, after the father speaks, the children are in a state of knowledge S characterized by the fact that every child knows both that $\mathbf{m}$ holds and that S holds. Thus, S satisfies the equation

$$S \equiv E(\mathbf{m} \wedge S).$$

The fixed point axiom of Section 6 says that under a view-based interpretation, $C_G\varphi$ is a solution for X in an analogous fixed point equation, namely

$$X \equiv E_G(\varphi \wedge X).$$

Now this equation has many solutions, including, for example, both *false* and $C_G(\varphi \wedge \psi)$, for any formula ψ. $C_G\varphi$ can be characterized as being the *greatest fixed*

point of the equation; that is, a fixed point that is implied by all other solutions. (The *least fixed point* of this equation is *false*, since it implies all other solutions.) As our discussion of common knowledge in the case of the muddy children puzzle suggests, expressing common knowledge as a greatest fixed point of such an equation seems to correspond more closely to the way it actually arises. We sketch a semantics for a propositional view-based logic of knowledge with fixed point in Appendix A. This alternative point of view, considering common knowledge as the greatest fixed point of such an equation, will turn out to be very useful when we attempt to define related variants of common knowledge.

11. ϵ-*Common Knowledge and* $\diamond$-*Common Knowledge*

Since, strictly speaking, common knowledge cannot be attained in practical distributed systems, it is natural to ask what states of knowledge *can* be obtained by the communication process. In this section, we consider what states of knowledge are attained in systems in which communication delivery is guaranteed but message delivery times are uncertain. For ease of exposition, we restrict our attention to view-based interpretations of knowledge here and in the next section.

We begin by considering *synchronous broadcast* channels of communication; that is, ones in which every message sent is received by all processors, and there are constants L and ϵ such that all processors receive the message between L and $L + \epsilon$ time units from the time it is sent. We call ϵ the *broadcast spread* of such a channel. Recall that the properties of the system hold throughout all of its runs and hence are common knowledge. In particular, the properties of the broadcast channel are common knowledge under any view-based interpretation.

Let us now consider the state of knowledge of the system when a processor p_i receives a broadcast message m. Clearly p_i knows that within an interval of ϵ time units around the current time everyone (receives m and) knows $sent(m)$. But p_i also knows that any other processor that receives m will know that all processors will receive m within such an ϵ interval. Let us define *within an* ϵ *interval, everyone knows* φ, denoted $E^{\epsilon}\varphi$, to hold if there is an interval of ϵ time units containing the current time such that each processor comes to know φ at some point in this interval. Formally, we have: $(\mathcal{S}, r, t) \models E_G^{\epsilon}\varphi$ if there exists an interval $I = [t', t' + \epsilon]$ such that $t \in I$ and for all $p_i \in G$ there exists $t_i \in I$ for which $(\mathcal{S}, r, t_i) \models K_i\varphi$. Let ψ be "some processor has received m." In a synchronous broadcast system as described above, we clearly have that $\psi \supset E^{\epsilon}\psi$ is valid.

We are thus in a state of knowledge that is analogous to common knowledge; here, however, rather than everyone knowing φ at the same instant, they all come to know φ within an interval of ϵ time units. We call this the state of group knowledge ϵ-common knowledge, denoted C^{ϵ}. The formal definition of $C_G^{\epsilon}\varphi$ is as the greatest fixed point of the equation:

$$X \equiv E_G^{\epsilon}(\varphi \wedge X).$$

We refer the reader to Appendix A for a rigorous definition. The fact ψ above, stating that some processor received the message m, has the property that $\psi \supset C^{\epsilon}\psi$. In addition, since $\psi \supset sent(m)$ is valid, it is also the case that $\psi \supset C^{\epsilon}sent(m)$. Thus, when some processor receives m it becomes ϵ-common knowledge that m has been sent.

As a straightforward consequence of its definition, C^{ϵ} satisfies the appropriate analogues of the fixed point axiom C1 and the induction rule C2 of Section 6 (replacing E by E^{ϵ} and C by C^{ϵ}). Note that we did not define $C_G^{\epsilon}\varphi$ as an infinite

conjunction of $(E_G^\epsilon)^k\varphi$, $k \geq 1$. Although it is not hard to show that $C_G^\epsilon\varphi$ implies this infinite conjunction, it is not equivalent to it; however, giving a detailed counterexample is beyond the scope of this paper. (We give an example of a similar phenomenon below.) The fixed-point definition is the one that is appropriate for our applications. Just as common knowledge corresponds to simultaneous actions in a distributed system, ϵ-common knowledge corresponds to actions that are guaranteed to be performed within ϵ time units of one another. This is what we get from the fixed-point axiom C1, which does not hold in general for the infinite conjunction. We are often interested in actions that are guaranteed to be performed within a small time window. For example, in an "early stopping" protocol for Byzantine agreement (cf. [8]), all correct processors are guaranteed to decide on a common value within ϵ time units of each other. It follows that once the first processor decides, the decision value is ϵ-common knowledge.[5]

There is one important special case where it can be shown that the fixed-point definition of $C_G^\epsilon\varphi$ is equivalent to the infinite conjunction. This arises when we restrict attention to complete-history interpretations and *stable* facts, facts that once true, remain true. Many facts of interest in distributed systems applications, such as "φ held at some point in the past," "the initial value of x is 1," or "φ holds at time t on p_i's clock," are stable. If φ is stable, then it is not hard to check that in complete-history interpretations, we have that $E_G^\epsilon\varphi$ holds iff $E_G\varphi$ will hold in ϵ time units. As a straightforward consequence of this observation, we can show that in complete-history interpretations, for a stable fact φ, we do have that $C_G^\epsilon\varphi$ holds iff $(E_G^\epsilon)^k\varphi$ holds for all $k \geq 1$.[6]

It is not hard to verify that of the properties of S5, C^ϵ satisfies only A3 (positive introspection) and R1 (the rule of necessitation). The failure of C^ϵ to satisfy the knowledge axiom and the consequence closure axiom can be traced to the failure of E^ϵ to satisfy these axioms. The problem is that $E^\epsilon\varphi$ only requires that φ hold and be known at some (not all!) of the points in the ϵ interval I. Indeed, it is not hard to construct an example in which $E^\epsilon\varphi \wedge E^\epsilon\neg\varphi$ holds. We remark that if we restrict attention to stable facts and complete-history interpretations, then consequence closure does hold for both E^ϵ and C^ϵ.

It is interesting to compare ϵ-common knowledge with common knowledge. Clearly, $C\varphi \supset C^\epsilon\varphi$ is valid. However, since synchronous broadcast channels are implementable in systems where common knowledge is not attainable, the converse does not hold. Thus, ϵ-common knowledge is strictly weaker than common knowledge. Moreover, note that while $C\varphi$ is a static state of knowledge, which can be true of a point in time irrespective of its past or future, $C^\epsilon\varphi$ is a notion that is essentially temporal. Whether or not it holds depends on what processors will know in an ϵ interval around the current time.

For any message m broadcast on a channel with broadcast spread ϵ, the fact $sent(m)$ becomes ϵ-common knowledge L times units after m is broadcast (in particular, as soon as it is sent if $L = 0$). Upon receiving m, a processor p_i knows that $C^\epsilon sent(m)$ holds, that is, $K_iC^\epsilon sent(m)$ holds. Returning to R2 and D2's communication problem, we can view them as a synchronous broadcast system,

[5] The situation there is in fact slightly more complicated since only the correct processors are required to decide; see [36] for definitions of knowledge appropriate for such situations.

[6] We remark that in earlier versions of this paper, we restricted attention to complete-history interpretations and stable facts, and defined $E_G^\epsilon\varphi$ as $\bigcirc^\epsilon E_G\varphi$, where $\bigcirc^\epsilon\psi$ is true at a point (r, t) iff ψ is true ϵ time units later, at $(r, t + \epsilon)$. By the comments above, our current definition is a generalization of our former definition.

and indeed they attain $C^{\epsilon}sent(m)$ immediately when R2 sends the message m. (Note that $L = 0$ in this particular example; the interested reader is invited to check that R2 and D2 in fact achieve $\epsilon/2$-common knowledge of $sent(m)$ at time $t_S + \epsilon/2$.)

Having discussed states of knowledge in synchronous broadcast channels, we now turn our attention to systems in which communication is *asynchronous*: no bound on the delivery times of messages in the system exists. Consider the state of knowledge of $sent(m)$ in a system in which m is broadcast over an *asynchronous channel*: a channel that guarantees that every message broadcast will *eventually* reach every processor. Upon receiving m, a processor knows $sent(m)$, and knows that every other processor either has already received m or will eventually receive m. This situation, where it is common knowledge that if m is sent then everyone will eventually know that m has been sent, gives rise to a weak state of group knowledge that we call *eventual common knowledge.*

We define *everyone in G will eventually have known* φ, denoted $E_G^{\diamond}\varphi$, to hold if for every processor in G there is some time during the run at which it knows φ. Formally, $(\mathcal{S}, r, t) \models E_G^{\diamond}\varphi$ if for all $p_i \in G$ there exists $t_i \geq 0$ such that $(\mathcal{S}, r, t_i) \models K_i\varphi$. We remark that if we restrict attention to stable facts φ and complete-history interpretations, then $E_G^{\diamond}\varphi$ is equivalent to $\diamond E_G\varphi$, that is, eventually everyone in G knows φ.[7] We define the state of $\diamond$-*common knowledge* (read *eventual* common knowledge), denoted by $C^{\diamond}$, by taking $C_G^{\diamond}\varphi$ to be the greatest fixed point of the equation:

$$X \equiv E_G^{\diamond}(\varphi \wedge X).$$

Notice that we again used the fixed-point definition rather than one in terms of infinite conjunction of $(E_G^{\diamond})^k\varphi$, $k \geq 1$. Our definition implies the infinite conjunction but, as we show by example below, it is not equivalent to the infinite conjunction, even if we restrict to stable facts and complete-history interpretations.

Our motivation for considering the fixed-point definition is the same as it was in the case of ϵ-common knowledge. The fixed-point definition gives us analogues to C1 and C2; as a consequence, $\diamond$-common knowledge corresponds to events that are guaranteed to take place at all sites *eventually.* For example, in some of the work on variants of the Byzantine Agreement problem discussed in the literature (cf. [8]), the kind of agreement sought is one in which whenever a correct processor decides on a given value, each other correct processor is guaranteed to decide on the same value *eventually.* The state of knowledge of the decision value that the processors attain in such circumstances is $\diamond$-common knowledge. Also, in asynchronous error-free broadcast channels, a processor knows that $sent(m)$ is $\diamond$-common knowledge when it receives the message m.

$C_G^{\diamond}$ is the weakest temporal notion of common knowledge that we have introduced. In fact, we now have a hierarchy of the temporal notions of common knowledge. For any fact φ and $\epsilon_1 \leq \cdots \leq \epsilon_k \leq \epsilon_{k+1} \leq \cdots$, we have:

$$C_G\varphi \supset C_G^{\epsilon_1}\varphi\varphi \supset \cdots \supset C_G^{\epsilon_k}\varphi \supset C_G^{\epsilon_{k+1}}\varphi \supset \cdots \supset C_G^{\diamond}\varphi.$$

We next consider how C^{ϵ} and $C^{\diamond}$ are affected by communication not being guaranteed. Recall that Theorem 5 implies that if communication is not guaranteed, then common knowledge is unaffected by the communication process. A fact only

[7] Formally, we take $\diamond\psi$ to be true at a point (r, t) if ψ is true at some point (r, t') with $t' \geq t$. In an earlier version of this paper, we defined $E_G^{\diamond}\varphi$ as $\diamond E_G\varphi$. Again, by the comments above, our current definition is a generalization of our former one.

becomes common knowledge if it becomes common knowledge in the absence of messages. Interestingly, the obvious analogue of Theorem 5 does not hold for C^ϵ and $C^\diamond$. Indeed, it is possible to construct a situation in which $C^\epsilon\varphi$ is attained *only* if communication is not sufficiently successful. For example, consider a system consisting of R2 and D2 connected by a two-way link. Communication along the link is not guaranteed, R2 and D2's clocks are perfectly synchronized, and both of them run the following protocol:

At time 0, *send the message "OK". For all natural numbers* $k > 0$, *if you have received* k *"OK" messages by time* k *on your clock, send an "OK" message at time* k; *otherwise, send nothing.*

Let ψ = "it is time k where $k \geq 1$ and some message sent at or before time $k - 1$ was not delivered within one time unit." Assume a complete-history interpretation for this system and fix $\epsilon = 1$. It is easy to see that $\psi \supset E^\epsilon\psi$ is valid in this system. For suppose that at time k the fact ψ holds because one of R2's messages was not delivered to D2. D2 knows ψ at time k and, according to the protocol, will not send a message to R2 at time k. Thus, by time $k + 1$, R2 will also know ψ (if it did not know it earlier). The induction rule implies that $\psi \supset C^\epsilon\psi$ is also valid in the system. If r is a run of the system where no messages are received, then it is easy to see that ψ holds at $(r, 1)$, and hence so does $C^\epsilon\psi$. However, $C^\epsilon\psi$ does not hold at $(r', 1)$ if r' is a run where all messages are delivered within one time unit. (The same example works for $C^\diamond\psi$.)

In the example above, successful communication in a system where communication is not guaranteed can prevent $C_G^\epsilon\psi$ (resp., $C_G^\diamond\psi$) from holding. However, the following theorem shows that we can get a partial analogue to Theorem 5 for C^ϵ and $C^\diamond$. Intuitively, it states that if $C_G^\epsilon\psi$ (resp., $C_G^\diamond\psi$) does not hold in the absence of successful communication, then $C_G^\epsilon\psi$ (resp., $C_G^\diamond\psi$) does not hold regardless of how successful communication may turn out to be. More formally,

THEOREM 9. *Let* R *and* G *be as in Theorem* 5, *and let* $\mathcal{I}$ *be a view-based interpretation. Let* r^- *be a run of* R *where no messages are received. If* $(\mathcal{I}, r^-, t) \not\models C_G^\epsilon\varphi$ (*resp.*, $(\mathcal{I}, r^-, t) \not\models C_G^\diamond\varphi$) *for all times* t, *then* $(\mathcal{I}, r, t) \not\models C_G^\epsilon\varphi$ (*resp.*, $(\mathcal{I}, r, t) \not\models C_G^\diamond\varphi$) *for all runs* r *with the same initial configuration and the same clock readings as* r^- *and all times* t.

PROOF. We sketch the proof for $C_G^\epsilon\varphi$; the proof for $C_G^\diamond\varphi$ is analogous. We assume that all runs mentioned in this proof have the same initial configuration and the same clock readings as r^-. If r is a run such that $C_G^\epsilon\varphi$ holds at some point in r, let $t_j(r)$ be the first time in r that processor $p_j \in G$ knows $C_G^\epsilon\varphi$. Let $\hat{t}(r) = \max\{t_j(r): p_j \in G\}$, and let $d(r)$ be the number of messages that are received in r up to (but not including) $\hat{t}(r)$. We show by induction on k that if r is a run such that $C_G^\epsilon\varphi$ holds at some point in r, then $d(r) \neq k$. This will show that in fact $C_G^\epsilon\varphi$ can never hold.

If $d(r) = 0$ and $C_G^\epsilon\varphi$ holds at some point in r, choose some $p_i \in G$ and let $t_i = t_i(r)$. Then we have that $(\mathcal{I}, r, t_i) \models K_iC_G^\epsilon\varphi$. Clearly $h(p_i, r, t_i) = h(p_i, r^-, t_i)$, so $(\mathcal{I}, r^-, t_i) \models K_iC_G^\epsilon\varphi$. By the knowledge axiom, we have that $(\mathcal{I}, r^-, t_i) \models C_G^\epsilon\varphi$, contradicting the hypothesis of the theorem.

For the inductive step, assume that $d(r) = k + 1$ and let $\hat{t} = \hat{t}(r)$. We now proceed as in the proof of Theorem 5. Let p_j be a processor receiving the last message received in r before time $\hat{t}$. Let t' be the time at which p_j receives this message. Let p_i be a processor in G such that $p_i \neq p_j$ and let $t_i = t_i(r)$. Since communication is not guaranteed, there exists a run r' extending (r, t') such that

(1) no messages are received in r' at or after time $\hat{t}$, (2) $h(p_i, r, t'') = h(p_i, r', t'')$ for all $t'' \leq \hat{t}$, and (3) all processors $p_k \neq p_i$ receive no messages in the interval $[t', \hat{t})$. By construction, at most k messages are received altogether in r', so $d(r') \leq k$. By the induction hypothesis we have that $(\mathcal{I}, r', t'') \models \neg C_G^\epsilon \varphi$ for all t''. It follows that $(\mathcal{I}, r', t_i) \models \neg K_i C_G^\epsilon \varphi$. But since we assumed $(\mathcal{I}, r, t_i) \models K_i C_G^\epsilon \varphi$ and $h(p_i, r, t_i) = h(p_i, r', t_i)$, this gives us a contradiction. □

We can now use Theorem 9 to prove an analogue to Corollary 6, which shows that if communication is not guaranteed, then there is no protocol for eventually coordinated attack.

PROPOSITION 10. *In the coordinated attack problem, any protocol that guarantees that whenever either party attacks the other party will eventually attack, is a protocol in which necessarily neither party attacks.*

PROOF. The proof is analogous to that of Corollary 6. Assume that (P_A, P_B) is a joint protocol that guarantees that if either party attacks then they both eventually attack, and let R be the corresponding system. Let $\psi =$ "*At least one of the generals has started attacking*". We first show that when either general attacks, then eventual common knowledge of ψ must hold. Since the protocol guarantees that whenever one general attacks the other one eventually attacks, it is easy to see that a general that has decided to attack knows ψ and knows that eventually both generals will know ψ. Thus, by the induction rule for $C^\diamond$, when a general attacks $C^\diamond \psi$ holds. Since in every run of the protocol in which no messages are received no party attacks (and hence neither ψ nor $C^\diamond \psi$ hold in such runs), by Theorem 9, the protocol (P_A, P_B) guarantees that neither general will ever attack. □

Theorem 9 allows us to construct an example in which the infinite conjunction of $(E^\diamond)^k \varphi$ holds, but $C^\diamond \varphi$ does not. In the setting of the coordinated attack problem, let φ be "*General A is in favor of attacking*". Consider a run in which all messengers arrive safely, and messages are acknowledged ad infinitum. Clearly, assuming a complete-history interpretation, for all k it is the case that $E^k \varphi$ holds after the kth message is delivered. It follows that $(E^\diamond)^k \varphi$ holds at time 0. However, Theorem 9 implies that $C^\diamond \varphi$ never holds in this run. It follows that $C^\diamond \varphi$ is not equivalent to the infinite conjunction of $(E^\diamond)^k \varphi$ even in the case of stable facts φ and complete-history interpretations.

Recall that the proof that unreliable communication cannot affect what facts are common knowledge carried over to (reliable) asynchronous communication. Our proof in Theorem 9 clearly does not carry over. In fact, a message broadcast over a reliable asynchronous channel *does* become eventual common knowledge. However, it is possible to show that asynchronous channels cannot be used in order to attain ϵ-common knowledge:

THEOREM 11. *Let R be a system with unbounded delivery times and let $|G| \geq 2$. Suppose there is some run r^- in R in which no messages are delivered in the interval $[0, t + \epsilon)$ such that $(\mathcal{I}, r^-, t) \not\models C_G^\epsilon \psi$. Then for all runs r in R with the same initial configuration and the same clock readings as r^-, we have $(\mathcal{I}, r, t) \not\models C_G^\epsilon \psi$.*

SKETCH OF PROOF. The proof essentially follows the proof of Theorems 5 and 9. We proceed by induction on $d(r)$, the number of messages received in r up to time t. Details are left to the reader. □

Thus, asynchronous communication channels are of no use for coordinating actions that are guaranteed to be performed at all sites within a predetermined fixed time bound.

12. *Timestamping: Using Relativistic Time*

Real time is not always the appropriate notion of time to consider in a distributed system. Processors in a distributed system often do not have access to a common source of real time, and their clocks do not show identical readings at any given real time. Furthermore, the actions taken by the processors rarely actually depend on real time. Rather, time is often used mainly for correctly sequencing events at the different sites and for maintaining "consistent" views of the state of the system. In this section, we consider states of knowledge relative to *relativistic* notions of time.

Consider the following scenario: R2 knows that R2 and D2's clock differ by at most δ, and that any message R2 sends D2 will arrive within ϵ time units. R2 sends D2 the following message m':

> "This message is being sent at t_S on R2's clock, and will reach D2 by $t_S + \epsilon + \delta$ on both clocks; m."

Let us denote $t_S + \epsilon + \delta$ by T_0. Now, at time T_0 on his clock, R2 would like to claim that $sent(m')$ is common knowledge. Is it? Well, we know by now that it is not, but it is interesting to analyze this situation. Before we do so, let us introduce a relativistic formalism for knowledge, which we call *timestamped* knowledge: We denote "at time T on his clock, p_i knows φ" by $K_i^T\varphi$. T is said to be the *timestamp* associated with this knowledge. We then define

$$E_G^T\varphi \equiv \bigwedge_{p_i \in G} K_i^T\varphi.$$

$E^T\varphi$ corresponds to everyone knowing φ individually at time T on their own clocks. Notice that for T_0 as above, $sent(m') \supset E^{T_0} sent(m')$. It is natural to define the corresponding relativistic variant of common knowledge, C^T, which we call *timestamped* common knowledge, so that $C_G^T\varphi$ is the greatest fixed point of the equation

$$X \equiv E_G^T(\varphi \wedge X).$$

So, in any run where the message m' is sent, R2 and D2 have timestamped common knowledge of $sent(m')$ with timestamp T_0. It is easy to check that C^T satisfies the fixed point axiom and the induction rule, as well as all of the axioms of S5 except for the knowledge axiom. In this respect, C^T resembles C more closely than C^ϵ and $C^\diamond$ do.

It is interesting to investigate how the relativistic notion of timestamped common knowledge relates to the notions of common knowledge, ϵ-common knowledge, and $\diamond$-common knowledge. Not surprisingly, the relative behavior of the clocks in the system plays a crucial role in determining the meaning of C^T.

THEOREM 12. *For any fact φ and view-based interpretation,*

(*a*) *if it is guaranteed that all clocks show identical times, then at time T on any processor's clock, $C_G^T\varphi \equiv C_G\varphi$.*

(*b*) *if it is guaranteed that all clocks are within ϵ time units of each other, then at time T on any processor's clock, $C_G^T\varphi \supset C_G^\epsilon\varphi$.*

(c) *if it is guaranteed that each local clock reads T at some time, then* $C_G^T\varphi \supset C_G^\diamond\varphi$.

Theorem 12 gives conditions under which C^T can be replaced by C, C^ϵ, and $C^\diamond$. A weak converse of Theorem 12 holds as well. Suppose the processors are able to set their clocks to a commonly agreed upon time T when they come to know $C_G\varphi$ (resp., come to know $C_G^\epsilon\varphi$, $C_G^\diamond\varphi$). Then it is easy to see that whenever $C_G\varphi$ (resp. $C_G^\epsilon\varphi$, $C_G^\diamond\varphi$) is attainable, so is $C_G^T\varphi$.

In many distributed systems, timestamped common knowledge seems to be a more appropriate notion to reason about than "true" common knowledge. Although common knowledge cannot be attained in practical systems, timestamped common knowledge is attainable in many cases of interest and seems to correspond closely to the relevant phenomena with which protocol designers are confronted. For example, in distributed protocols that work in phases, we speak of the state of the system at the beginning of phase 2, at the end of phase k, and so on. It is natural to think of the phase number as a "clock" reading, and consider knowledge about what holds at the different phases as "timestamped" knowledge, with the phase number being the timestamp. In certain protocols for Byzantine agreement, for example, the nonfaulty processors attain common knowledge of the decision value at the end of phase k (cf. [9, 36]). In practical systems in which the phases do not end simultaneously at the different sites of the system, the processors can be thought of as actually attaining timestamped common knowledge of the decision value, with the timestamp being "the end of phase k." Indeed, protocols like the atomic broadcast protocol of [6] are designed exactly for the purpose of attaining timestamped common knowledge. (See [38] for more discussion of timestamped common knowledge.)

13. *Internal Knowledge Consistency*

We have seen that common knowledge closely corresponds to the ability to perform *simultaneous* actions. In the last few sections, we introduced a number of related states of knowledge corresponding to weaker forms of coordinated actions. Such weaker forms of coordination are often sufficient for many practical applications. This helps explain the paradox of the happy existence of practical distributed systems despite the apparent need for common knowledge and the negative results of Theorem 8.

However, there are situations where we act as if—or we would like to carry out our analysis as if—we had true common knowledge, not a weaker variant. For example, in the muddy children puzzle, even though simultaneity may not be attainable, we want to assume that the children do indeed have common knowledge of the father's statement. As another example, consider a protocol that proceeds in phases, in which it is guaranteed that no processor will ever receive a message out of phase. In many cases, all the aspects of this protocol that we may be interested in are faithfully represented if we model the system as if it were truly synchronous: All processors switch from one phase to the next simultaneously.

Intuitively, in both cases, the assumption of common knowledge seems to be a *safe* one, even if it is not quite true. We would like to make this intuition precise. Recall that an epistemic interpretation is one that specifies what a processor believes at any given point as a function of the processor's history at that point. An epistemic interpretation $\mathcal{I}$ is a knowledge interpretation if it is *knowledge consistent*, that is, if it has the property that whenever $(\mathcal{I}, r, t) \models K_i\varphi$ then also $(\mathcal{I}, r, t) \models \varphi$. Now an epistemic interpretation that is not knowledge consistent

may nevertheless be *internally knowledge consistent*, which intuitively means that the processors never obtain information from within the system that would contradict the assumption that the epistemic interpretation is in fact a knowledge interpretation. In other words, no processor ever has information that implies that the knowledge axiom $K_i\varphi \supset \varphi$ is violated. More formally, an epistemic interpretation $\mathcal{I}$ for a system R is said to be *internally knowledge consistent* if there is a subsystem $R' \subseteq R$ such that $\mathcal{I}$ is a knowledge interpretation when restricted to R', and for all processors p_i and points (r, t) of R, there is a point (r', t') in R' such that $h(p_i, r, t) = h(p_i, r', t')$.

Given that epistemic interpretations ascribe knowledge (or, perhaps more appropriately in this case, beliefs) to processors as a function of the processors' histories, the above definition implies that whenever a processor is ascribed knowledge of a certain fact at a point of R, then as far as any events involving this processor at the current and at any future time are concerned, it is consistent to assume that the fact does indeed hold.

Using the notion of internal knowledge consistency, we can make our previous intuitions precise. When analyzing the muddy children puzzle, we assume that the children will never discover that they did not hear and comprehend the father's statement simultaneously. We take the set R' from the definition of internal knowledge consistency here to be precisely the set of runs where they did hear and comprehend the father's statement simultaneously. Similarly, in the case of the protocol discussed above, the set R' is the set where all processors advance from one phase to the next truly simultaneously. It now also makes sense to say that under reasonable conditions processors can safely use an "eager" protocol corresponding to the eager epistemic interpretation of Section 8, in which processors act as if they had common knowledge, even though common knowledge does not hold. It is possible to give a number of conditions on the ordering of events in the system that will ensure that it will be internally knowledge consistent for the processors to act as if they have common knowledge.

For further discussion on internal knowledge consistency, see the recent paper by Neiger [37].

14. *Conclusions*

In this paper, we have tried to bring out the important role of reasoning about knowledge in distributed systems. We have shown that reasoning about the knowledge of a group and its evolution can reveal subtleties that may not otherwise be apparent, can sharpen our understanding of basic issues, and can improve the high-level reasoning required in the design and analysis of distributed protocols and plans.

We introduced a number of states of group knowledge, but focused much of our attention on one particular state, that of common knowledge. We showed that, in a precise sense, common knowledge is a prerequisite for agreement. However, we also showed that in many practical systems common knowledge is not attainable. This led us to consider three variants of common knowledge—ϵ-common knowledge, eventual common knowledge, and timestamped common knowledge—that are attainable in practice, and may suffice for carrying out a number of actions. The methodology we introduce for constructing these variants of common knowledge, involving the fixed-point operator, can be used to construct other useful variants of common knowledge. Indeed, recent papers have introduced *concurrent common knowledge* [39], *probabilistic common knowledge* [12], and *polynomial-time common knowledge* [34], using this methodology.

There is clearly much more work to be done in terms of gaining a better understanding of knowledge in distributed systems. This paper considers a general model of a distributed system. It would also be useful to consider knowledge in distributed systems with particular properties. The work of Chandy and Misra [4] is an interesting study of this kind (see [9], [13], and [18] for other examples). We carried out a knowledge-based analysis of the coordinated attack problem here. Since this paper first appeared, a number of other problems, including Byzantine agreement, distributed commitment, and mutual exclusion, have been analyzed in terms of knowledge (see [4], [9], [18], [24], [31], [36], and [38]). Such knowledge-based analyses both shed light on the problem being studied and improve our understanding of the methodology. More studies of this kind would further deepen our understanding of the issues involved.

Another general direction of research is that of using knowledge for the specification and verification of distributed systems. (See [26] for an initial step in this direction.) Formalisms based on knowledge may prove to be a powerful tool for specifying and verifying protocols, and may also be readily applicable to the synthesis of protocols and plans. Temporal logic has already proved somewhat successful in this regard [10, 30].

Our analysis of the muddy children puzzle and the coordinated attack problem, as well as the work in [9], [20], [35], and [36] illustrate how subtle the relationship between knowledge, action, and communication in a distributed system can be. In this context, Halpern and Fagin (cf. [20]) look at *knowledge-based protocols*, which are protocols in which a processor's actions are explicitly based on the processor's knowledge. This provides an interesting generalization of the more standard notions of protocols.

In the long run, we hope that a theory of knowledge, communication, and action will prove rich enough to provide general foundations for a unified theoretical treatment of distributed systems. Such a theory also promises to shed light on aspects of knowledge that are relevant to related fields.

Appendix A

In this appendix we present a logic with a greatest fixed-point operator and illustrate how common knowledge and variants of common knowledge can be formally defined as greatest fixed points. Our presentation follows that of Kozen [27].

Intuitively, given a system R, a formula ψ partitions the points of R into two sets: those that satisfy ψ, and those that do not. We can identify a formula with the set of points that satisfy it. In order to be able to define fixed points of certain formulas, which is our objective in this appendix, we consider formulas that may contain a free variable whose values range over subsets of the points of R. Once we assign a set of points to the free variable, the formula can be associated with a set of points in a straightforward way (as will be shown below). Thus, such a formula can be viewed as a function from subsets of R to subsets of R. (A formula with no free variable is then considered a constant function, yielding the same subset regardless of the assignment.)

Before we define the logic more formally, we need to review a number of relevant facts about fixed points. Suppose S is a set and f is a function mapping subsets of S to subsets of S. A subset A of S is said to be a *fixed point* of f if $f(A) = A$. A *greatest* (resp., *least*) fixed point of f is a set B such that $f(B) = B$, and if $f(A) = A$, then $A \subseteq B$ (resp., $B \subseteq A$). It follows that if f has a greatest fixed point B, then $B = \bigcup \{A : f(A) = A\}$. The function f is said to be *monotone increasing* if $f(A) \subseteq$

$f(B)$ whenever $A \subseteq B$ and *monotone decreasing* if $f(A) \supseteq f(B)$ whenever $A \subseteq B$. The Knaster–Tarski theorem (cf. [43]) implies that a monotone increasing function has a greatest (and a least) fixed point. Given a function f and a subset A, define $f^0(A) = A$ and $f^{i+1}(A) = f(f^i(A))$. f is said to be *downward continuous* if $f(\bigcap_i A_i) = \bigcap_i f(A_i)$ for all sequences $A_1, A_2, \ldots$ with $A_1 \supseteq A_2 \supseteq \cdots$. Given a monotone increasing and downward continuous function f it is not hard to show that the greatest fixed point of f is the set $\bigcap_{k<\omega} f^k(S)$. We remark that if f is monotone increasing but not downward continuous, then we can still obtain the greatest fixed points of f in this fashion, but we have to extend the construction by defining f^α for all ordinals α.[8]

We are now in a position to formally define our logic. We start with a set $\Phi = \{P, Q, P_1, \ldots\}$ of primitive propositions and a single propositional variable X. We form more complicated formulas by allowing the special formula *true* and then closing off under conjunction, negation, the modal operators K_i, E_G, E_G^ϵ, and $E_G^\diamond$ for every group G of processors, and the greatest fixed-point operator νX. Thus, if φ and ψ are formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $K_i\varphi$, $E_G\varphi$, $E_G^\epsilon\varphi$, $E_G^\diamond\varphi$, and $\nu X.\varphi$ (read "the greatest fixed point of φ with respect to X"). However, we place a syntactic restriction, described below, on formulas of the form $\nu X.\varphi$.

Just as $\forall x$ in first-order logic binds occurrences of x, νX binds occurrences of X. Thus, in a formula such as $X \wedge \neg E_G^\epsilon(\nu X.[X \wedge (K_1X \wedge K_2X)])$, the first occurrence of X is free, while the rest are bound. We say that a free occurrence of X in a formula φ is *positive* if it is in the scope of an even number of negation signs, and *negative* if it is in the scope of an odd number of negation signs. Thus, in a formula such as $X \wedge \neg K_1X$, the first occurrence of X is positive while the second is negative. The restriction on formulas of the form $\nu X.\varphi$ is that all free occurrences of X in φ must be positive; the point of this restriction will be explained below.

The next step is to associate with each formula a function. Given a distributed system represented by its set of runs R, let $S = R \times [0, \infty)$. A model $\mathcal{M}$ is a triple (S, π, v), where S is as above, π associates a truth assignment to the primitive propositions with each point in S, and v: $\{1, \ldots, m\} \times S \rightarrow \Sigma$ is an assignment of views (from a set of states Σ) to the processors at the points of S. We now associate with each formula φ a function $\varphi^{\mathcal{M}}$ from subsets of S to subsets of S. Intuitively, if no occurrences of X are free in φ, then $\varphi^{\mathcal{M}}$ will be a constant function, and $\varphi^{\mathcal{M}}(A)$ will be the set of points where φ is true (no matter how we choose A). If X is free in φ, then $\varphi^{\mathcal{M}}(A)$ is the set of points where φ is true if A is the set of points where X is true. We define $\varphi^{\mathcal{M}}(A)$ by induction on the structure of φ as follows:

(a) $X^{\mathcal{M}}(A) = A$ (so $X^{\mathcal{M}}$ is the identity function).
(b) $P^{\mathcal{M}}(A) = \{s \in S: \pi(s)(P) = \textbf{true}\}$ for a primitive proposition P.
(c) $true^{\mathcal{M}}(A) = S$.
(d) $(\neg\varphi)^{\mathcal{M}}(A) = S - \varphi^{\mathcal{M}}(A)$.
(e) $(\varphi \wedge \psi)^{\mathcal{M}}(A) = \varphi^{\mathcal{M}}(A) \cap \psi^{\mathcal{M}}(A)$.
(f) $(K_i\varphi)^{\mathcal{M}}(A) = \{(r, t) \in S$: for all $(r', t') \in S$, $v(p_i, r, t) = v(p_i, r', t')$ implies $(r', t') \in \varphi^{\mathcal{M}}(A)\}$.
(g) $(E_G\varphi)^{\mathcal{M}}(A) = \bigcap_{i \in G}(K_i\varphi)^{\mathcal{M}}(A)$.
(h) $(E_G^\epsilon\varphi)^{\mathcal{M}}(A) = \{(r, t) \in S$: there exists an interval $I = [t', t' + \epsilon]$ with $t \in I$, such that $\forall p_i \in G\ \exists t_i \in I((r, t_i) \in (K_i\varphi)^{\mathcal{M}}(A))\}$.

[8] We can similarly define a function f to be *upward continuous* if $f(\bigcup_i A_i) = \bigcup_i f(A_i)$ for all sequences $A_1, A_2, \ldots$ with $A_1 \subseteq A_2 \subseteq \cdots$. For monotone increasing upward continuous functions f, the least fixed point of f is $\bigcup_{k<\omega} f^k(\emptyset)$. Again, to get least fixed points in the general case, we have to extend this construction through the ordinals.

(i) $(E_G^{\Diamond}\varphi)^{\mathcal{M}}(A) = \{(r, t) \in S: \forall p_i \in G\ \exists t_i((r, t_i) \in (K_i\varphi)^{\mathcal{M}}(A))\}$.
(j) $(\nu X.\varphi)^{\mathcal{M}}(A) = \bigcup\{B: \varphi^{\mathcal{M}}(B) = B\}$.

Now by an easy induction on the structure of formulas we can prove the following facts:

(1) If φ is a formula in which all free occurrences of X are positive (resp., negative), then $\varphi^{\mathcal{M}}$ is monotone increasing (resp., monotone decreasing). Note that our syntactic restriction on formulas of the form $\nu X.\varphi$ guarantees that for a well-formed formula of this form, the function $\varphi^{\mathcal{M}}$ is monotone increasing. As a consequence, $(\nu X.\varphi)^{\mathcal{M}}(A)$ is the greatest fixed point of the function $\varphi^{\mathcal{M}}$.
(2) If φ is a formula with no free variables, then $\varphi^{\mathcal{M}}$ is a constant function. In particular, observe that $(\nu X.\varphi)^{\mathcal{M}}$ is necessarily a constant function (the definition shows that $(\nu X.\varphi)^{\mathcal{M}}(A)$ is independent of the choice of A). As well, it is easy to check that if φ is a valid formula such as $\neg(P \wedge \neg P)$, then $\varphi^{\mathcal{M}}(A) = S$.
(3) For formulas in which the variable X does not appear (so, in particular, for formulas not involving the greatest fixed point operator), $\varphi^{\mathcal{M}}(A) = \{(r, t): (\mathcal{I}_v, r, t) \models \varphi\}$, where $\mathcal{I}_v$ is the view-based interpretation associated with the view function v. (Again, this is true for any choice of A, since by the previous observation, $\varphi^{\mathcal{M}}$ is a constant function if there is no occurrence of X in φ.) Thus, if we define $(\mathcal{M}, r, t) \models \varphi$ iff $(r, t) \in \varphi^{\mathcal{M}}(\varnothing)$, then this definition extends our previous definition (in that for formulas in which the variable X does not appear, we have $(\mathcal{M}, r, t) \models \varphi$ iff $(\mathcal{I}_v, r, t) \models \varphi$).

Given the machinery at our disposal, we can now formally define $C_G\varphi$ as $\nu X.E_G(\varphi \wedge X)$, define $C_G^{\varepsilon}\varphi$ as $\nu X.E_G^{\varepsilon}(\varphi \wedge X)$, and define $C_G^{\Diamond}\varphi$ as $\nu X.E_G^{\Diamond}(\varphi \wedge X)$. It follows from our characterization of greatest fixed points of downward continuous functions that if $\varphi^{\mathcal{M}}$ is downward continuous, then $\nu X.\varphi$ is equivlaent to $\varphi_0 \wedge \varphi_1 \wedge \cdots$, where φ_0 is *true*, φ_{i+1} is $\varphi[\varphi_i/X]$, and $\varphi[\psi/X]$ denotes the result of substituting ψ for the free occurrences of X in φ. It is easy to check that $(E_G(\varphi \wedge X))^{\mathcal{M}}$ is downward continuous if $\varphi^{\mathcal{M}}$ is downward continuous. In particular, if φ has no free occurrences of X (so that $\varphi^{\mathcal{M}}$ is constant), it follows that we have:

$$C_G\varphi \equiv E_G\varphi \wedge E_G(\varphi \wedge E_G\varphi) \wedge E_G(\varphi \wedge E_G(\varphi \wedge E_G\varphi)) \cdots.^{9}$$

Since $E_G(\psi_1 \wedge \psi_2) \equiv (E_G\psi_1 \wedge E_G\psi_2)$ it follows that

$$C_G\varphi \equiv E_G\varphi \wedge E_GE_G\varphi \wedge \cdots.$$

However, $(E_G^{\varepsilon}(\varphi \wedge X))^{\mathcal{M}}$ and $(E_G^{\Diamond}(\varphi \wedge X))^{\mathcal{M}}$ are *not* necessarily downward continuous. The reason that $(E_G^{\Diamond}(\varphi \wedge X))^{\mathcal{M}}$ is not downward continuous is that an infinite collection of facts can each *eventually* hold, without them necessarily all holding simultaneously at some point. We have already seen one example of this phenomenon in Section 11. For another example, suppose we are working in a system with an unbounded global clock, and let $A_i = (\mathit{current\_time} > i)^{\mathcal{M}}$. Since the clock is unbounded, it follows that $A_i \neq \varnothing$ for all i, but $\bigcap_i A_i = \varnothing$. Taking ψ to be the formula $E^{\Diamond}(\varphi \wedge X)$, it is easy to see that $(r, 0) \in \psi^{\mathcal{M}}(A_i)$ for all i, and hence $\bigcap_i(\psi^{\mathcal{M}}(A_i)) \neq \psi^{\mathcal{M}}(\bigcap_i A_i)$.

[9] Note that the formula on the right-hand side of the equivalence is not in our language, since we have not allowed infinite conjunctions. However, we can easily extend the language to allow infinite conjunctions in the obvious way so that the equivalence holds.

We can construct a similar example in the case of E^ϵ, because we have taken time to range over the reals. For example, if we take x_i to be an infinite sequence of real numbers converging from below to ϵ, take $A_i = (\mathit{current\_time} \in (x_i, \epsilon))^{\mathcal{N}}$, and now take ψ to be the formula $E^\epsilon(\varphi \wedge X)$, then again we have $\bigcap_i A_i = \emptyset$, and $(r, 0) \in \bigcap_i (\psi^{\mathcal{N}}(A_i))$. This example does depend crucially on the fact that time ranges over the reals. If instead we had taken time to range over the natural numbers, then we would in fact get downward continuity.

We encourage the reader to check that $C_G\varphi$, $C_G^\epsilon\varphi$, and $C_G^\diamond\varphi$ all satisfy the fixed-point axiom and the induction rule. The fixed-point axiom is a special case of the more general fixed-point axiom $\nu X.\varphi \equiv \varphi[\nu X.\varphi/X]$, while the induction rule is a special case of the more general induction rule for fixed points: from $\psi \supset \varphi[\psi/X]$ infer $\psi \supset \nu X.\varphi$. The reader might also now wish to check that C has the properties of S5, while C^ϵ and $C^\diamond$ satisfy the positive introspection axiom and the necessitation rule. Furthermore, for stable facts and complete-history interpretations, they also satisfy the consequence closure axiom. C^ϵ and $C^\diamond$ satisfy neither the knowledge axiom nor the negative introspection axiom. We remark that both notions satisfy weaker variants of the knowledge axiom: $C^\epsilon\varphi$ implies that φ holds at some point at most ϵ time units away from the current point, while $C^\diamond\varphi$ implies that φ holds (at least) at some point during the run.

It is straightforward to extend the above framework to include explicit individual clock times in order to define $C_G^T\varphi$ (see [38] for more details). Here, for example, it is the case that $(E^T(\varphi \wedge X))^{\mathcal{N}}$ is downward continuous, and E^T distributes over conjunction; hence C^T will coincide with the appropriate infinite conjunction. Similar treatments can be applied to many related variants of common knowledge (see, e.g., [12], [34], and [39]).

Appendix B

In this appendix we fill in the details of the proof that common knowledge cannot be attained in practical systems (Theorem 8 in Section 8).

Our first step is to establish a general condition—namely, that the initial point of a run is reachable from any later point—under which common knowledge can be neither gained nor lost. We remark that Chandy and Misra have shown that in the case of completely asynchronous, event-driven systems where communication is not guaranteed, common knowledge of any fact can be neither gained nor lost [4]. Since it is easy to see that, in such systems, the initial point of a run is reachable from all later points, our result provides a generalization of that of [4].

PROPOSITION 13. *Let $r \in R$ be a run in which the point $(r, 0)$ is G-reachable from (r, t) in the graph corresponding to the complete-history interpretation, and let $\mathcal{I}$ be a knowledge interpretation for R. Then for all formulas φ we have $(\mathcal{I}, r, t) \models C_G\varphi$ iff $(\mathcal{I}, r, 0) \models C_G\varphi$.*

PROOF. Fix a run r, time t, and formula φ. Since $(r, 0)$ is G-reachable from (r, t) in the graph corresponding to the complete-history interpretation, there exist points $(r_0, t_0), (r_1, t_1), \ldots, (r_k, t_k)$ such that $(r, t) = (r_0, t_0)$, $(r, 0) = (r_k, t_k)$, and for every $i < k$ there is a processor $j_i \in G$ that has the same history at (r_i, t_i) and at (r_{i+1}, t_{i+1}). We can now prove by induction on i, using Lemma 2, that $(\mathcal{I}, r, t) \models C_G\varphi$ iff $(\mathcal{I}, r_i, t_i) \models C_G\varphi$. The result follows. □

We next provide a formal definition of systems with temporal imprecision, and show that in such systems, the initial point of a run is always reachable from later

points. A system R has *temporal imprecision* if

$$\forall r \in R \quad \forall t \geq 0 \quad \forall i \forall j \neq i \quad \exists \delta > 0 \quad \forall \delta' \in [0, \delta) \quad \exists r' \forall t' < t$$
$$(h(p_i, r, t') = h(p_i, r', t' + \delta') \wedge h(p_j, r, t') = h(p_j, r', t'))).$$

Intuitively, this means that processors cannot perfectly coordinate their notions of time in a system with temporal imprecision. One processor might always be a little behind the others.

By *reachable* in the following lemma we mean reachable (in the sense of Section 6) with respect to the view function defined by the complete-history interpretation.

LEMMA 14. *If R is a system with temporal imprecision, then for all runs $r \in R$ and times t, the point $(r, 0)$ is reachable from (r, t).*

PROOF. Let R be a system with temporal imprecision and (r, t) be a point of R. Suppose $t \neq 0$ (otherwise, clearly $(r, 0)$ is reachable from (r, t)). Let t_0 be the greatest lower bound of the set $\{t' : (r, t'')$ is reachable from (r, t) for all $t'' \in [t', t]\}$. We show that (r, t_0) is reachable from (r, t) and that $t_0 = 0$. Since R is a system with temporal imprecision, there exists a δ such that for all δ' with $0 < \delta' < \delta$, there exists a run r' such that for all $t' \leq t$, we have $h(p_1, r, t') = h(p_1, r', t' + \delta')$ and $h(p_i, r, t') = h(p_i, r', t')$ for $i \neq 1$. If $\delta' < t' \leq t$, it follows that (r', t') is reachable from (r, t') and $(r, t' - \delta')$ is reachable from (r', t'). By transitivity of reachability, we have that $(r, t' - \delta')$ is reachable from (r, t'), and by symmetry, that (r, t') is reachable from $(r, t' - \delta')$. It now follows that $(r, t - \delta')$ is reachable from (r, t) for all $\delta' < \min(\delta, t)$. Thus $t_0 \leq t - \min(\delta, t)$. Furthermore, if $\delta' < \min(\delta, t)$, then we know that $(r, t_0 + \delta')$ is reachable from both (r, t_0) and (r, t). It thus follows that (r, t_0) is reachable from (r, t). Finally, if $t_0 \neq 0$, then we know that $(r, t_0 - \delta')$ is reachable from (r, t_0) (and hence from (r, t)) for all $\delta' < \min(t_0, \delta)$. But this contradicts our choice of t_0. Thus, $t_0 = 0$, and $(r, 0)$ is reachable from (r, t). □

Theorem 8 now follows as an immediate corollary to Lemma 14 and Proposition 13.

We conclude by showing that many practical systems do indeed have temporal imprecision (although the δ's involved in some cases might be very small). Perhaps through statistical data, we can assume that for every communication link l there are known lower and upper bounds L_l and H_l, respectively, on the message delivery time for messages over l. We assume that the message delivery time on the link l is always in the open interval (L_l, H_l). (We take the interval to be open here since it seems reasonable to suppose that if the system designer considers it possible that a message will take time T to be delivered, then for some sufficiently small $\delta > 0$, he will also consider it possible that the delivery time is anywhere in the interval $(T - \delta, T + \delta)$; in this we differ slightly from [7] and [21]. We define f_l to be a *message delivery function for link l* if $f_l : \mathbf{N} \to (L_l, H_l)$. A run r is *consistent* with f_l if for all $n \in \mathbf{N}$, $f_l(n)$ is the delivery time of the nth message in r on link l. A system R has *bounded but uncertain message delivery times* if for all links l there exist bounds $L_l < H_l$ such that for all runs $r \in R$ and all message delivery functions $f_l : \mathbf{N} \to (L_l, H_l)$, there exists a run r' that is identical to r except that message delivery time over the link l is defined by f_l. More formally, r' is consistent with f_l and for all i, processor p_i follows the same protocol, wakes up at the same time (i.e., $t_{\text{init}}(p_i, r) = t_{\text{init}}(p_i, r')$), and has the same initial state and the same clock readings in both r and r'.

We say R is a system with *uncertain start times* if there exists $\delta_0 > 0$ such that given a run $r \in R$, a processor p_i, and δ with $0 < \delta < \delta_0$, there is a run r' which is identical to r except that p_i wakes up δ earlier in r' with its clock readings (if there are clocks in the system) shifted back by δ. More formally, for all $j \neq i$, processor p_j follows the same protocol, wakes up at the same time, and has the same initial state in both r and r'. Moreover, for all k, the delivery time for the kth message on link l (if there is one) is the same in both r and r'. All processors other than p_i have the same clock readings in both r and r'. Processor p_i starts δ later in r' than r, although it has the same initial state in both runs, and $\tau(p_i, r, t) = \tau(p_i, r', t + \delta)$.

For any practical system, it seems reasonable to assume that there will be some (perhaps very small) uncertainty in start times and, even if message delivery is guaranteed within a bounded time, that there is some uncertainty in message delivery time. These assumptions are sufficient to guarantee temporal imprecision, as the following result, whose proof is a slight modification of a result proved in [7] on the tightness of clock synchronization achievable, shows:

PROPOSITION 15. *A system with bounded but uncertain message delivery times and uncertain start times has temporal imprecision.*

SKETCH OF PROOF. Let (r, t) be a point of the system, and let p_i be a processor. Let δ_0 be as in the definition of uncertain start times. Since only a finite number of messages are received by time t in r, there is some $\delta > 0$ such that the delivery times of these messages are more than δ greater than the lower bound for the particular link they were sent over, and more than δ less than the upper bound. Choose $\delta' < \min(\delta_0, \delta)$ and some processor p_i. Let r' be a run in which all processors $p_j \neq p_i$ start at the same time and in the same initial state as in r, have the same clock readings (if there are clocks), and all messages between such processors take exactly the same time as in r. In addition, processor p_i starts δ' time units later in r' than in r, messages to p_i take δ' time units longer to be delivered, while messages from p_i are delivered δ' time units faster than in r, and p_i's clock readings (if there are clocks) are shifted by δ'. Such a run r' exists by our assumptions. It is not hard to check that run r' has the property that for all times $t' \leq t$, all processors $p_j \neq p_i$ have exactly the same history at time t' in both r and r', while processor p_i has the same history at (r, t') and at $(r', t' + \delta')$. Since (r, t) and p_i were chosen arbitrarily, it thus follows that the system has temporal imprecision. □

ACKNOWLEDGMENTS. This work evolved from work the authors did with Danny Dolev on [35]. Many people commented on different versions of this work. Of special value were comments by Dave Chelberg, Steve Deering, Cynthia Dwork, Ron Fagin, Vassos Hadzilacos, Danny Lehmann, Yoni Malachi, Tim Mann, Andres Modet, Gil Neiger, Jan Pachl, Derek Proudian, Stan Rosenschein, Yoav Shoham, Ray Strong, Moshe Vardi, Joe Weening, and Lenore Zuck. Jan Pachl suggested the term *distributed knowledge* to replace the term *implicit knowledge* that we have been using. We would particularly like to thank Gil Neiger and Lenore Zuck for an outstanding job of refereeing, well beyond the call of duty.

REFERENCES

1. AUMANN, R. J. Agreeing to disagree. *Ann. Stat. 4*, 6 (1976), 1236–1239.
2. BARWISE, J. Scenes and other situations. *J. Philo. LXXVIII* 7 (1981), 369–397.

3. CHANDY, K. M., AND LAMPORT, L. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst. 3*, 1 (1985), 63–75.
4. CHANDY, K. M., AND MISRA, J. How processes learn. *Dist. Comput. 1*, 1 (1986), 40–52.
5. CLARK, H. H., AND MARSHALL, C. R. Definite reference and mutual knowledge. In *Elements of Discourse Understanding*, A. K. Joshi, B. L. Webber, and I. A. Sag, eds. Cambridge University Press, Cambridge, Mass., 1981, pp. 10–63.
6. CRISTIAN, F., AGHILI, H., STRONG, H. R., AND DOLEY, D. Atomic broadcast: From simple diffusion to Byzantine agreement. In *Proceedings of the 15th International Annual Symposium on Fault-Tolerant Computing Systems*. IEEE, Washington, D.C., 1985, pp. 200–206.
7. DOLEV, D., HALPERN, J. Y., AND STRONG, H. R. On the possibility and impossibility of achieving clock synchronization. *J. Comput. Syst. Sci. 32*, 2 (1986), 230–250.
8. DOLEV, D., REISCHUK, R., AND STRONG, H. R. Eventual is earlier than immediate. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*. IEEE, Washington, D.C., 1982, pp. 196–203.
9. DWORK, C., AND MOSES, Y. Knowledge and common knowledge in a Byzantine environment I: Crash failures (extended abstract). In *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, J. Y. Halpern, ed. Morgan Kaufmann, San Mateo, Calif., 1986, pp. 149–170. *Inf. Computation*, in press.
10. EMERSON, E. A., AND CLARKE, E. M. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Prog. 2* (1982), 241–266.
11. FAGIN, R., AND HALPERN, J. Y. Belief, awareness, and limited reasoning. *Artif. Int. 34* (1988), 39–76.
12. FAGIN, R., AND HALPERN, J. Y. Reasoning about knowledge and probability: Preliminary report. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*, M. Y. Vardi, ed. Morgan Kaufmann, San Mateo, Calif., 1988, pp. 277–293.
13. FAGIN, R., HALPERN, J. Y., AND VARDI, M. Y. What can machines know? On the epistemic properties of machines. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, 1986, pp. 428–434. A revised and expanded version appears as: What can machines know? On the properties of knowledge in distributed systems. IBM Res. Rep. RJ56250. IBM, 1988.
14. FISCHER, M. J., AND IMMERMAN, N. Foundations of knowledge for distributed systems. In *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, J. Y. Halpern, ed. Morgan Kaufmann, San Mateo, Calif., 1986, pp. 171–186.
15. FISCHER, M. J., LYNCH, N. A., AND PATERSON, M. S. Impossibility of distributed consensus with one faulty processor. *J. ACM 32*, 2 (1985), 374–382.
16. GALLAGER, R. G. Seminar on computer communication networks. Office of Industrial Liaison, MIT, Cambridge, Mass., 1979.
17. GRAY, J. Notes on database operating systems. IBM Res. Rep. RJ 2188. IBM, Aug. 1987.
18. HADZILACOS, V. A knowledge-theoretic analysis of atomic commitment protocols. In *Proceedings of the 6th ACM Symposium on Principles of Database Systems*. ACM, New York, 1987, pp. 129–134. A revised version has been submitted for publication.
19. HALPERN, J. Y. Using reasoning about knowledge to analyze distributed systems. In *Annual Review of Computer Science*, Vol. 2, J. Traub, B. Grosz, B. Lampson, and N. Nilson, eds. Annual Reviews, Inc., Palo Alto, Calif., 1987, pp. 37–68.
20. HALPERN, J. Y., AND FAGIN, R. A formal model of knowledge, action, and communication in distributed systems: Preliminmary report. In *Proceedings of the 4th ACM Symposium on Principles of Distributed Computing*. ACM, New York, 1985, pp. 224–236. A revised version appears as: Modelling knowledge and action in distributed systems. *Dist. Computations 3* (1989), 159–177.
21. HALPERN, J. Y., MEGIDDO, N., AND MUNSHI, A. Optimal precision in the presence of uncertainty. *J. Complexity 1* (1985), 170–196.
22. HALPERN, J. Y., AND MOSES, Y. Knowledge and common knowledge in a distributed environment. In *Proceedings of the 3rd ACM Conference on Distributed Computing*. ACM, New York, 1984, pp. 50–61. A revised and expanded version appears as: IBM Res. Rep. RJ 4421. IBM, Yorktown Heights, N.Y., 1988.
23. HALPERN, J. Y., AND MOSES, Y. A guide to the modal logics of knowledge and belief. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85)*. 1985, pp. 480–490.
24. HALPERN, J. Y., AND ZUCK, L. D. A little knowledge goes a long way: Simple knowledge-based derivations and correctness proofs for a family of protocols. IBM Res. Rep. RJ 5857. IBM, 1987.
25. HINTIKKA, J. *Knowledge and Belief*. Cornell University Press, Ithaca, N.Y., 1962.
26. KATZ, S., AND TAUBENFELD, G. What processes know: Definitions and proof methods. In *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing*. ACM, New York, 1986, pp. 249–262.

27. KOZEN, D. Results on the propositional μ-calculus. *Theoret. Comput. Sci. 27*, 1 (1983), 333–354.
28. LADNER, R., AND REIF, J. The logic of distributed protocols (preliminary report). In *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, J. Y. Halpern, ed. Morgan Kaufman, San Mateo, Calif., 1986, pp. 207–222.
29. LEVESQUE, H. A logic of implicit and explicit belief. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*. 1984, pp. 198–202.
30. MANNA, Z., AND WOLPER, P. L. Synthesis of communication processes from temporal logic specifications. *ACM Trans. Program. Lang. Syst. 6*, 1 (1984), 68–93.
31. MAZER, M. S. A knowledge theoretic account of recovery in distributed systems: The case of negotiated commitment. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*, M. Y. Vardi, ed. Morgan Kaufman, San Mateo, Calif., 1988, pp. 309–324.
32. MCCARTHY, J., SATO, M., HAYASHI, T., AND IGARISHI, S. On the model theory of knowledge. Tech. Rep. STAN-CS-78-657. Stanford Univ., Stanford, Calif., 1979.
33. MOORE, R. C. A formal theory of knowledge and action. In *Formal Theories of the Commonsense World*, J. Hobbs and R. C. Moore, eds. Ablex Publishing Corp., Norwood, N.J., 1985, pp. 319–358.
34. MOSES, Y. Resource-bounded knowledge. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*, M. Y. Vardi, ed. Morgan Kaufmann, San Mateo, Calif., 1988, pp. 261–276.
35. MOSES, Y., DOLEV, D., AND HALPERN, J. Y. Cheating husbands and other stories: A case study of knowledge, action, and communication. *Dist. Comput. 1*, 3 (1986), 167–176.
36. MOSES, Y., AND TUTTLE, M. R. Programming simultaneous actions using common knowledge. *Algorithmica 3* (1988), 121–169.
37. NEIGER, G. Knowledge consistency: A useful suspension of disbelief. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*, M. Y. Vardi, ed. Morgan Kaufman, San Mateo, Calif., 1988, pp. 295–308.
38. NEIGER, G., AND TOUEG, S. Substituting for real time and common knowledge in asynchronous distributed systems. *J. ACM*.
39. PANANGADEN, P., AND TAYLOR, S. Concurrent common knowledge: a new definition of agreement for asynchronous systems. In *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing*. ACM, New York, 1988, pp. 197–209.
40. PARIKH, R., AND RAMANUJAM, R. Distributed processing and the logic of knowledge. In *Proceedings of the Workshop on Logics of Programs*, R. Parikh, ed. Springer-Verlag, Berlin, 1985, pp. 256–268.
41. ROSENSCHEIN, S. J. Formal theories of AI in knowledge and robotics. *New Gen. Comput. 3* (1985), 345–357.
42. ROSENSCHEIN, S. J., AND KAELBLING, L. P. The synthesis of digital machines with provable epistemic properties. In *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, J. Y. Halpern, ed. Morgan Kaufmann, San Mateo, Calif., 1986, pp. 83–97.
43. TARSKI, A. A lattice-theoretic fixpoint theorem and its applications. *Pacific J. Math. 5* (1955), 285–309.
44. YEMINI, Y., AND COHEN, D. Some issues in distributed processes communication. In *Proceedings of the 1st International Conference on Distributed Computing Systems*. IEEE, Washington, D.C., 1979, pp. 199–203.

RECEIVED OCTOBER 1984; REVISED NOVEMBER 1985, AUGUST 1987, AND NOVEMBER 1988; ACCEPTED JUNE 1989

Solution of a Problem in Concurrent Programming Control

E. W. DIJKSTRA
Technological University, Eindhoven, The Netherlands

A number of mainly independent sequential-cyclic processes with restricted means of communication with each other can be made in such a way that at any moment one and only one of them is engaged in the "critical section" of its cycle.

Introduction

Given in this paper is a solution to a problem for which, to the knowledge of the author, has been an open question since at least 1962, irrespective of the solvability. The paper consists of three parts: the problem, the solution, and the proof. Although the setting of the problem might seem somewhat academic at first, the author trusts that anyone familiar with the logical problems that arise in computer coupling will appreciate the significance of the fact that this problem indeed can be solved.

The Problem

To begin, consider N computers, each engaged in a process which, for our aims, can be regarded as cyclic. In each of the cycles a so-called "critical section" occurs and the computers have to be programmed in such a way that at any moment only one of these N cyclic processes is in its critical section. In order to effectuate this mutual exclusion of critical-section execution the computers can communicate with each other via a common store. Writing a word into or nondestructively reading a word from this store are undividable operations; i.e., when two or more computers try to communicate (either for reading or for writing) simultaneously with the same common location, these communications will take place one after the other, but in an unknown order.

The solution must satisfy the following requirements.

(a) The solution must be symmetrical between the N computers; as a result we are not allowed to introduce a static priority.

(b) Nothing may be assumed about the relative speeds of the N computers; we may not even assume their speeds to be constant in time.

(c) If any of the computers is stopped well outside its critical section, this is not allowed to lead to potential blocking of the others.

(d) If more than one computer is about to enter its critical section, it must be impossible to devise for them such finite speeds, that the decision to determine which one of them will enter its critical section first is postponed until eternity. In other words, constructions in which "After you"-"After you"-blocking is still possible, although improbable, are not to be regarded as valid solutions.

We beg the challenged reader to stop here for a while and have a try himself, for this seems the only way to get a feeling for the tricky consequences of the fact that each computer can only request one one-way message at a time. And only this will make the reader realize to what extent this problem is far from trivial.

The Solution

The common store consists of:

"**Boolean array** b, $c[1{:}N]$; **integer** k"

The integer k will satisfy $1 \leq k \leq N$, $b[i]$ and $c[i]$ will only be set by the ith computer; they will be inspected by the others. It is assumed that all computers are started well outside their critical sections with all Boolean arrays mentioned set to **true**; the starting value of k is immaterial.

The program for the ith computer ($1 \leq i \leq N$) is:

```
"integer j;
Li0: b[i] := false;
Li1: if k ≠ i then
Li2: begin c[i] := true;
Li3: if b[k] then k := i;
     go to Li1
     end
       else
Li4: begin c[i] := false;
       for j := 1 step 1 until N do
         if j ≠ i and not c[j] then go to Li1
     end;
     critical section;
     c[i] := true; b[i] := true;
     remainder of the cycle in which stopping is allowed;
     go to Li0"
```

The Proof

We start by observing that the solution is safe in the sense that no two computers can be in their critical section simultaneously. For the only way to enter its critical section is the performance of the compound statement $Li4$ without jumping back to $Li1$, i.e., finding all other c's **true** after having set its own c to **false**.

The second part of the proof must show that no infinite "After you"-"After you"-blocking can occur; i.e., when none of the computers is in its critical section, of the computers looping (i.e., jumping back to $Li1$) at least one—and therefore exactly one—will be allowed to enter its critical section in due time.

If the kth computer is not among the looping ones, $b[k]$ will be **true** and the looping ones will all find $k \neq i$. As a result one or more of them will find in $Li3$ the Boolean $b[k]$ **true** and therefore one or more will decide to assign "$k := i$". After the first assignment "$k := i$", $b[k]$ becomes **false** and no new computers can decide again to assign a new value to k. When all decided assignments to k have been performed, k will point to one of the looping computers and will not change its value for the time being, i.e., until $b[k]$ becomes **true**, viz., until the kth computer has completed its critical section. As soon as the value of k does not change any more, the kth computer will wait (via the compound statement $Li4$) until all other c's are **true**, but this situation will certainly arise, if not already present, because all other looping ones are forced to set their c **true**, as they will find $k \neq i$. And this, the author believes, completes the proof.

Reaching Agreement in the Presence of Faults

M. PEASE, R. SHOSTAK, AND L. LAMPORT

SRI International, Menlo Park, California

ABSTRACT. The problem addressed here concerns a set of isolated processors, some unknown subset of which may be faulty, that communicate only by means of two-party messages. Each nonfaulty processor has a private value of information that must be communicated to each other nonfaulty processor. Nonfaulty processors always communicate honestly, whereas faulty processors may lie The problem is to devise an algorithm in which processors communicate their own values and relay values received from others that allows each nonfaulty processor to infer a value for each other processor The value inferred for a nonfaulty processor must be that processor's private value, and the value inferred for a faulty one must be consistent with the corresponding value inferred by each other nonfaulty processor

It is shown that the problem is solvable for, and only for, $n \geq 3m + 1$, where m is the number of faulty processors and n is the total number. It is also shown that if faulty processors can refuse to pass on information but cannot falsely relay information, the problem is solvable for arbitrary $n \geq m \geq 0$. This weaker assumption can be approximated in practice using cryptographic methods

KEY WORDS AND PHRASES. agreement, authentication, consistency, distributed executive, fault avoidance, fault tolerance, synchronization, voting

CR CATEGORIES: 3.81, 4.39, 5.29, 5.39, 6.22

1. *Introduction*

Fault-tolerant systems often require a means by which independent processors or processes can arrive at an exact mutual agreement of some kind. It may be necessary, for example, for the processors of a redundant system to synchronize their internal clocks periodically. Or they may have to settle upon a value of a time-varying input sensor that gives each of them a slightly different reading. In the absence of faults reaching a satisfactory mutual agreement is usually an easy matter. In most cases it suffices simply to exchange values (times, in the case of clock synchronization) and compute some kind of average. In the presence of faulty processors, however, simple exchanges cannot be relied upon; a bad processor might report one value to a given processor and another value to some other processors, causing each to calculate a different "average."

One might imagine that the effects of faulty processors could be dealt with through the use of voting schemes involving more than one round of information exchange; such schemes might force faulty processors to reveal themselves as faulty or at least to behave consistently enough with respect to the nonfaulty processors to allow the latter to reach an exact agreement. As we will show, it is not always possible to devise schemes of this kind, even if it is known that the faulty processors are in a minority. Algorithms that allow exact agreement to be reached by the nonfaulty processors do exist, however, if they sufficiently outnumber the faulty ones.

Our results are formulated using the notion of *interactive consistency*, which we define

This work was supported in part by NASA-Langley Research Center under Contract NASI-13792 and by the Ballistic Missile Defense Advanced Technology Center under Contract DASG60-78-C-0046

Authors' address Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025

as follows: Consider a set of n isolated processors, of which it is known that no more than m are faulty. It is not known, however, which processors are faulty. Suppose that the processors can communicate only by means of two-party messages. The communication medium is presumed to be fail-safe and of negligible delay. The sender of a message, moreover, is always identifiable by the receiver. Suppose also that each processor p has some private value of information V_p (such as its clock value or its reading of some sensor). The question is whether for given m, $n \geq 0$, it is possible to devise an algorithm based on an exchange of messages that will allow each nonfaulty processor p to compute a vector of values with an element for each of the n processors, such that

(1) the nonfaulty processors compute exactly the same vector;
(2) the element of this vector corresponding to a given nonfaulty processor is the private value of that processor.

Note that the algorithm need not reveal which processors are faulty, and that the elements of the computed vector corresponding to faulty processors may be arbitrary; it matters only that the nonfaulty processors compute exactly the same value for any given faulty processor.

We say that such an algorithm achieves interactive consistency, since it allows the nonfaulty processors to come to a consistent view of the values held by all the processors, including the faulty ones. The computed vector is called an *interactive consistency* (*i.c.*) *vector*. Once interactive consistency has been achieved, each nonfaulty processor can apply an averaging or filtering function to the i.c. vector, according to the needs of the application. Since each nonfaulty processor applies this function to the same vector of values, an exact agreement is necessarily reached.

We show in the following sections that algorithms can be devised to guarantee interactive consistency for and only for n, m such that $n \geq 3m + 1$. It will follow, in particular, that a minimum of four processors is required in the single-fault case. We also show, however, that interactive consistency can be assured for arbitrary $n \geq m \geq 0$ if it is assumed that faulty processors can refuse to pass on information obtained from other processors but cannot falsely report this information. This assumption can be approximated in practice using *authenticators*, which we discuss in Section 5.

We begin in Section 2 with a description of the single-fault case. Section 3 is concerned with the generalization to $n \geq 3m + 1$ and Section 4 with an impossibility argument for $n \leq 3m$. Section 5 gives an algorithm for arbitrary $n \geq m \geq 0$ that works under the restricted assumption stated above. Conclusions and issues for future study are given in Section 6.

Problems similar to the one considered here have been studied by Davies and Wakerly [1].

2. *The Single-Fault Case*

In order to give the reader a feeling for the problem, we begin with a procedure for obtaining interactive consistency in the simple case of $m = 1$, $n = 4$. The procedure consists of an exchange of messages, followed by the computation of the interactive consistency vector on the basis of the results of the exchange.

Two rounds of information exchange are required. In the first round the processors exchange their private values. In the second round they exchange the results obtained in the first round. The faulty processor (if there is one) may "lie," of course, or refuse to send messages. If a nonfaulty processor p fails to receive a message it expects from some other processor, p simply chooses a value at random and acts as if that value had been sent.

The exchange having been completed, each nonfaulty processor p records its private value V_p for the element of the interactive consistency corresponding to p itself. The element corresponding to every other processor q is obtained by examining the three received reports of q's value (one of these was obtained directly from q in the first round, and the others from the remaining two processors in the second round). If at least two of

the three reports agree, the majority value is used. Otherwise, a default value such as "NIL" is used.

To see that this procedure assures interactive consistency, first note that if q is nonfaulty, p will receive V_q both from q and from the other nonfaulty processor(s). Thus p will record V_q for q as desired. Now suppose q is faulty. We must show only that p and the other two nonfaulty processors record the same value for q. If every nonfaulty processor records NIL, we are done. Otherwise, some nonfaulty processor, say p, records a non-NIL value v, having received v from at least two other processors. Now if p received v from both of the other nonfaulty processors, each other nonfaulty processor must receive v from every processor other than p (and possibly from p as well); every nonfaulty processor will thus record v. Otherwise, p must have received v from all processors other than some other nonfaulty processor p'. In this case p' received v from all processors other than q (so p' records v), and all other nonfaulty processors received v from all processors other than p'. All nonfaulty processors therefore record v as required.

3. *A Procedure for* $n \geq 3m + 1$

Recall that the procedure given in the last section requires two rounds of information exchange, the first consisting of communications of the form "my private value is" and the second consisting of communications of the form "processor x told me his private value is ...." In the general case of m faults, $m + 1$ rounds are required. In order to describe the algorithm, it will be convenient to characterize this exchange of messages in a more formal way.

Let P be the set of processors and V a set of values. For $k \geq 1$, we define a *k-level scenario* as a mapping from the set of nonempty strings (possibly having repetitions) over P of length $\leq k + 1$, to V. For a given k-level scenario σ and string $w = p_1 p_2 \cdots p_r$, $2 \leq r \leq k + 1$, $\sigma(w)$ is interpreted as the value p_2 tells p_1 that p_3 told p_2 that p_4 told p_3 ... that p_r told p_{r-1} is p_r's private value. For a single-element string p, $\sigma(p)$ simply designates p's private value V_p. A k-level scenario thus summarizes the outcome of a k-round exchange of information. (Note that if a faulty processor lies about who gave it information, this is equivalent to lying about a value it was given.) Note also that for a given subset of nonfaulty processors, only certain mappings are possible scenarios; in particular, since nonfaulty processors are always truthful in relaying information, a scenario must satisfy

$$\sigma(pqw) = \sigma(qw)$$

for each nonfaulty processor q, arbitrary processor p, and string w.

The messages a processor p receives in a scenario σ are given by the restriction σ_p of σ to strings beginning with p. The procedure we present now for arbitrary $m \geq 0$, $n \geq 3m + 1$, is described in terms of p's computation, for a given σ_p, of the element of the interactive-consistency vector corresponding to each processor q. The computation is as follows:

(1) If for some subset Q of P of size $>(n + m)/2$ and some value v, $\sigma_p(pwq) = v$ for each string w over Q of length $\leq m$, p records v.

(2) Otherwise, the algorithm for $m - 1$, $n - 1$ is recursively applied with P replaced by $P - \{q\}$, and σ_p by the mapping $\hat{\sigma}_p$ defined by

$$\hat{\sigma}_p(pw) = \sigma_p(pwq)$$

for each string w of length $\leq m$ over $P - \{q\}$. If at least $\lfloor (n + m)/2 \rfloor$ of the $n - 1$ elements in the vector obtained in the recursive call agree, p records the common value, otherwise p records NIL.

Note that $\hat{\sigma}_p$ corresponds to the m-level subscenario of σ in which q is excluded and in which each processor's private value is the value it obtains directly from q in σ. Note also that the algorithm essentially reduces to the one given in the last section in the case $m = 1$, $n = 4$.

The proof that the algorithm given above does indeed assure interactive consistency proceeds by induction on m:

Basis $m = 0$. In this case no processor is faulty, and the algorithm always terminates in step (1) with p recording V_q for q.

Induction Step $m > 0$. First note that if q is nonfaulty, $\sigma_p(pwq) = V_q$ for each string w (including the empty string) of length $\leq m$ over the set of nonfaulty processors. This set has $n - m$ members (which, since $n > 3m$, is $>(n + m)/2$) and so satisfies the requirements for Q in step (1) of the algorithm. Any other set satisfying these requirements, moreover, must contain a nonfaulty processor (since it must be of size $>(n + m)/2$, and $n \geq 3m + 1$) and must therefore also yield V_q as the common value. The algorithm thus terminates at step (1), and p records V_q and q as required.

Now suppose that q is faulty. We must show that the value p records for q agrees with the value each other nonfaulty processor p' records for q.

First consider the case in which both p and p' exit the procedure at step (1), each having found an appropriate set Q. Since each such set has more than $(n + m)/2$ members, and since P has only n members in all, the two sets must have more than $2((n + m)/2) - n = m$ common members. Since at least one of these must be nonfaulty, the two sets must give rise to the same value v, as required.

Next suppose that p' exits at step (1), having found an appropriate set Q and common value v, and that p executes step (2). We claim that in the vector of $n - 1$ elements that p computes in the recursive call, the elements corresponding to members of $\hat{Q} = Q - \{q\}$ are equal to v. Since $\hat{Q}$ has at least $\lfloor (n + m)/2 \rfloor$ members, it will then follow that p records v in accordance with step (2). To see that the elements corresponding to members of $\hat{Q}$ are indeed equal to v, recall that the mapping $\hat{\sigma}_p$ that p uses to compute the vector in the recursive call is the restriction, to strings beginning with p, of the m-level scenario $\hat{\sigma}_p$ defined by

$$\hat{\sigma}_p(w) = \sigma(wq)$$

for each string w of length $\leq m$ over $P - \{q\}$. By the induction hypothesis, this vector is identical to the one p' would have computed using the restriction $\hat{\sigma}_{p'}$ of $\hat{\sigma}$ had p' made the recursive call. Moreover, the value p' would have computed for the element of this vector corresponding to a given q' in $\hat{Q}$ must be v, since $\hat{Q}$ and v satisfy step (1) of the algorithm. (Note that $\hat{Q}$ is of size $\geq \lfloor (n + m)/2 \rfloor > [(n - 1) + (m - 1)]/2$, and that $\sigma_{p'}(p'wq') = \sigma_{p'}(p'wq'q) = v$ for each string w of length $\leq m - 1$ over $\hat{Q}$.) The case in which p exits at step (1) and p' exits at step (2) is handled similarly.

In the one remaining case, both p and p' exit at step (2). In this case both recurse and must, by the induction hypothesis, compute exactly the same vector, and hence the same value for q. Q.E.D.

4. *Proof of Impossibility for* $n < 3m + 1$

The procedure of the last section guarantees interactive consistency only if $n \geq 3m + 1$. In this section it is shown that the $3m + 1$ bound is tight. We will prove not only that it is impossible to assure interactive consistency for $n < 3m + 1$ with $m + 1$ rounds of information exchange, but also that it is impossible, even allowing an infinite number of rounds of exchange (i.e., using scenarios mapping from *all* nonempty strings over P to V).

Just to gain some intuitive feeling as to why $3m$ processors are not sufficient, consider the case of three processors A, B, C, of which one, say C, is faulty. By prevaricating in just the right way, C can thwart A's and B's efforts to obtain consistency. In particular, C's messages to A can be such as to suggest to A that C's private value is, say, 1, and that B is faulty. Similarly, C's messages to B can be such as to suggest to B that C's private value is 2, and that A is faulty. If C plays its cards just right, A will not be able to tell whether B or C is faulty, and B will not be able to tell whether A or C is at fault. A will thus have no

choice but to record 1 for C's value, while B must record 2, defeating interactive consistency.

In order to give a precise statement of the impossibility result and its proof, a few formal definitions are needed.

First, define a *scenario* as a mapping from the set P^+ of all nonempty strings over P, to V. For a given $p \in P$ define a *p-scenario* as a mapping from the subset of P^+, consisting of strings beginning with p, to V.

Next, for a given choice $N \subseteq P$ of nonfaulty processors and a given scenario σ, say that σ is *consistent with* N if for each $q \in N$, $p \in P$, and $w \in P^*$ (set of all strings over P), $\sigma(pqw) = \sigma(qw)$. (In other words, σ is consistent with N if each processor in N always reports what it knows or hears truthfully.)

Now define the notion of interactive consistency as follows. For each $p \in P$, let F_p be a mapping that takes a p-scenario σ_p and a processor q as arguments and returns a value in V. (Intuitively, F_p gives the value that p computes for the element of the interactive consistency vector corresponding to q on the basis of σ_p.) We say that $\{F_p | p \in P\}$ assures *interactive consistency for m faults* if for each choice of $N \subseteq P$, $|N| \geq n - m$, and each scenario σ consistent with N,

(i) for all $p, q \in N$, $F_p(\sigma_p, q) = \sigma(q)$,
(ii) for all $p, q \in N$, $r \in P$, $F_p(\sigma_p, r) = F_q(\sigma_q, r)$,

where σ_p and σ_q denote the restrictions of σ to strings beginning with p and q, respectively.

Intuitively, clause (i) requires that each nonfaulty processor p correctly compute the private value of each nonfaulty processor q, and clause (ii) requires that each two nonfaulty processors compute exactly the same vector.

THEOREM. *If $|V| \geq 2$ and $n \geq 3m$, there exists no $\{F_p | p \in P\}$ that assures interactive consistency for m faults.*

PROOF. Suppose, to the contrary, that $\{F_p | p \in P\}$ assures interactive consistency for m faults. Since $n \leq 3m$, P can be partitioned into three nonempty sets A, B, and C, each of which has no more than m members. Let v and v' be two distinct values in V. Our general plan is to construct three scenarios α, β, and σ such that α is consistent with $N = A \cup C$, β with $N = B \cup C$, and σ with $N = A \cup B$. The members of C will all be given private value v in α and v' in β. Moreover, α, β, and σ will be constructed in such a way that no processor $a \in A$ can distinguish α from σ (i.e., $\alpha_a = \sigma_a$), and no processor $b \in B$ can distinguish β from σ (i.e., $\beta_b = \sigma_b$). It will then follow that for the scenario σ processors in A and B will compute different values for the members of C.

We define the scenarios α, β, and σ recursively as follows:

(i) For each $w \in P^+$ not ending in a member of C, let

$$\alpha(w) = \beta(w) = \sigma(w) = v.$$

(ii) For each $a \in A$, $b \in B$, $c \in C$ let

$$\begin{aligned} &\alpha(c) = \alpha(ac) = \alpha(bc) = \alpha(cc) = v, \\ &\beta(c) = \beta(ac) = \beta(bc) = \beta(cc) = v', \\ &\sigma(c) = \sigma(ac) = \sigma(cc) = v, \qquad \sigma(bc) = v'. \end{aligned}$$

(iii) For each $a \in A$, $b \in B$, $c \in C$, $p \in P$, $w \in P^*c$ (i.e., w is any string over P ending in c), let

$$\begin{aligned} &\alpha(paw) = \alpha(aw), \quad &&\beta(paw) = \alpha(aw), \\ &\alpha(pbw) = \beta(bw), \quad &&\beta(pbw) = \beta(bw), \\ &\alpha(pcw) = \alpha(cw), \quad &&\beta(pcw) = \beta(cw), \end{aligned}$$

$$\begin{aligned} \sigma(paw) &= \sigma(aw), \\ \sigma(pbw) &= \sigma(bw), \\ \sigma(acw) &= \alpha(cw), \\ \sigma(bcw) &= \beta(cw). \end{aligned}$$

It is easy to verify by inspection that α, β, and σ are in fact consistent with $N = A \cup C$, $B \cup C$, $A \cup B$, respectively. Moreover, one can show by a simple induction proof on the length of w that

$$\alpha(aw) = \sigma(aw), \qquad \beta(bw) = \sigma(bw)$$

for all $a \in A$, $b \in B$, and $w \in P^*$.

It then follows from the definition of interactive consistency that for any $a \in A$, $b \in B$, $c \in C$,

$$v = \alpha(c) = F_a(\alpha_a, c) = F_a(\sigma_a, c) = F_b(\sigma_b, c) = F_b(\beta_b, c) = v',$$

giving a contradiction. Q.E.D.

5. *An Algorithm Using Authenticators*

The negative result of the last section depends strongly on the assumption that a faulty processor may refuse to pass on values it has received from other processors or may pass on fabricated values. This section addresses the situation in which the latter possibility is precluded. We will assume, in other words, that a faulty processor may "lie" about its own value and may refuse to relay values it has received, but may not relay altered values without betraying itself as faulty.

In practice, this assumption can be satisfied to an arbitrarily high degree of probability using *authenticators*. An authenticator is a redundant augment to a data item that can be created, ideally, only by the originator of the data. A processor p constructs an authenticator for a data item d by calculating $A_p[d]$, where A_p is some mapping known only to p. It must be highly improbable that a processor q other than p can generate the authenticator $A_p[d]$ for a given d. At the same time, it must be easy for q to check, for a given p, v, and d, that $v = A_p[d]$. The problem of devising mappings with these properties is a cryptographic one. Methods for their constructions are discussed in [2] and [3]. For many applications in which faults are due to random errors rather than to malicious intelligence, any mappings that "suitably randomize" the data suffice.

A scenario σ is carried out in the following way. As before, let $v = \sigma(p)$ designate p's private value. p communicates this value to r by sending r the message consisting of the triple $\langle p, a, v \rangle$, where $a = A_p[v]$. When r receives the message, it checks that $a = A_p[v]$. If so, r takes v as the value of $\sigma(rp)$. Otherwise r lets $\sigma(rp) =$ NIL. More generally, if r receives exactly one message of the form $(p_1, a_1(p_2, a_2 \ldots (p_k, a_k, v) \ldots))$, where $a_k = A_k[v]$ and for $1 \leq i \leq k - 1$, $a_i = A_i[(p_{i+1}, a_{i+1} \ldots (p_k, a_k, v)]$, then $\sigma(rp_1 \cdots p_k) = v$. Otherwise, $\sigma(rp_1 \cdots p_k) =$ NIL.

A scenario σ constructed in this way is consistent with a given choice N of nonfaulty processors, if for all processors $p \in N$, $q \in P$ and strings w, w' over P.

(i) $\sigma(qpw) = \sigma(pw)$,
(ii) $\sigma(w'pw)$ is either $\sigma(pw)$ or NIL.

Condition (i) ensures that nonfaulty processors are always truthful. Condition (ii) guarantees that a processor cannot relay an altered value of information received from a nonfaulty processor.

We now present a procedure, using $m + 1$-level authenticated scenarios, that guarantees interactive consistency for any $n \geq m$. As before, the procedure is described in terms of the value a nonfaulty processor p records for a given processor q on the basis of σ_p:

> Let S_{pq} be the set of all non-NIL values $\sigma_p(pwq)$, where w ranges over strings of distinct elements with length $\leq m$ over $P - \{p, q\}$. If S_{pq} has exactly one element v, p records v for q; otherwise, p records NIL.

To see that interactive consistency is assured, consider first the case in which q is nonfaulty. In this case $\sigma_p(pwq)$ is either $\sigma(q)$ or NIL for each appropriate w by condition (ii). Since, in particular, $\sigma_p(pq) = \sigma(q)$ by (i), $S_{pq} = \{\sigma(q)\}$. p thus records $\sigma(q)$ for q as required.

If q is faulty, it suffices to show only that for each two nonfaulty processors p and p', $S_{pq} = S_{p'q}$. So suppose $v \in S_{pq}$, i.e., $v = \sigma_p(pwq)$ for some string w having no repetitions, with length $\leq m$ over $P - \{p, q\}$. If p' occurs in w (say $w = w_1p'w_2$), then $\sigma(pwq) = \sigma(p'w_2q)$ by (ii); hence $v = \sigma(pwq) \in S_{p'q}$. If p' does not occur in w and w is of length $<m$, then pw is of length $\leq m$; so $v = \sigma(pwq) = \sigma(p'pwq) \in S_{p'q}$. Finally, if p' does not occur in w and w is of length m, w must be of the form w_1rw_2 where r is nonfaulty, giving that $v = \sigma(pwq) = \sigma(rw_2q)$ (by (ii)) $= \sigma(p'rw_2q)$ (by (i)) $\in S_{p'q}$. In each case $v \in S_{p'q}$. A symmetrical argument shows that if $v \in S_{p'q}$, $v \in S_{pq}$. Hence $S_{p'q} = S_{pq}$ as required. Q.E.D.

6. *Conclusions*

The problem of obtaining interactive consistency appears to be quite fundamental to the design of fault-tolerant systems in which executive control is distributed. In the SIFT [4] fault-tolerant computer under development at SRI, the need for an interactive consistency algorithm arises in at least three aspects of the design—synchronization of clocks, stabilization of input from sensors, and agreement on results of diagnostic tests. In the preliminary stages of the design of this system, it was naively assumed that simple majority voting schemes could be devised to treat these situations. The gradual realization that simple majorities are insufficient led to the results reported here.

These results by no means answer all the questions one might pose about interactive consistency. The algorithms presented here are intended to demonstrate existence. The construction of efficient algorithms and algorithms that work under the assumption of restricted communications is a topic for future research. Other questions that will be considered include those of reaching approximate agreement and reaching agreement under various probabilistic assumptions.

ACKNOWLEDGMENTS. The authors gratefully acknowledge the substantial contribution of ideas to this paper by K. N. Levitt, P. M. Melliar-Smith, and J. H. Wensley, and the reviewers. We are especially grateful to E. Migneault of NASA-Langley for his perceptive insights into the importance and difficulty of the problem.

REFERENCES

1. DAVIES, D., AND WAKERLY, J. Synchronization and matching in redundant systems *IEEE Trans on Comptrs. C-27*, 6 (June 1978), 531–539.
2. DIFFIE, W., AND HELLMAN, M. New directions in cryptography. *IEEE Trans Inform. Theory IT-22*, 6 (Nov 1976), 644–654
3. RIVEST, R.L., SHAMIR, A., AND ADLEMAN, L A A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM 21*, 2 (Feb 1978), 120–126.
4. WENSLEY, J H., ET AL. SIFT: design and analysis of a fault-tolerant computer for aircraft control *Proc. IEEE 66*, 10 (Oct. 1978), 1240–1255.

RECEIVED NOVEMBER 1978; REVISED APRIL 1979; ACCEPTED MAY 1979

The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound two loyal generals. With unforgeable written messages, the problem is solvable for any number of generals and possible traitors. Applications of the solutions to reliable computer systems are then discussed.

Categories and Subject Descriptors: C.2.4. [**Computer-Communication Networks**]: Distributed Systems—*network operating systems*; D.4.4 [**Operating Systems**]: Communications Management—*network communication*; D.4.5 [**Operating Systems**]: Reliability—*fault tolerance*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Interactive consistency

1. INTRODUCTION

A reliable computer system must be able to cope with the failure of one or more of its components. A failed component may exhibit a type of behavior that is often overlooked—namely, sending conflicting information to different parts of the system. The problem of coping with this type of failure is expressed abstractly as the Byzantine Generals Problem. We devote the major part of the paper to a discussion of this abstract problem and conclude by indicating how our solutions can be used in implementing a reliable computer system.

We imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action. However, some of the generals

This research was supported in part by the National Aeronautics and Space Administration under contract NAS1-15428 Mod. 3, the Ballistic Missile Defense Systems Command under contract DASG60-78-C-0046, and the Army Research Office under contract DAAG29-79-C-0102.
Authors' address: Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025.

may be traitors, trying to prevent the loyal generals from reaching agreement. The generals must have an algorithm to guarantee that

A. All loyal generals decide upon the same plan of action.

The loyal generals will all do what the algorithm says they should, but the traitors may do anything they wish. The algorithm must guarantee condition A regardless of what the traitors do.

The loyal generals should not only reach agreement, but should agree upon a reasonable plan. We therefore also want to insure that

B. A small number of traitors cannot cause the loyal generals to adopt a bad plan.

Condition B is hard to formalize, since it requires saying precisely what a bad plan is, and we do not attempt to do so. Instead, we consider how the generals reach a decision. Each general observes the enemy and communicates his observations to the others. Let $v(i)$ be the information communicated by the ith general. Each general uses some method for combining the values $v(1), \ldots, v(n)$ into a single plan of action, where n is the number of generals. Condition A is achieved by having all generals use the same method for combining the information, and Condition B is achieved by using a robust method. For example, if the only decision to be made is whether to attack or retreat, then $v(i)$ can be General i's opinion of which option is best, and the final decision can be based upon a majority vote among them. A small number of traitors can affect the decision only if the loyal generals were almost equally divided between the two possibilities, in which case neither decision could be called bad.

While this approach may not be the only way to satisfy conditions A and B, it is the only one we know of. It assumes a method by which the generals communicate their values $v(i)$ to one another. The obvious method is for the ith general to send $v(i)$ by messenger to each other general. However, this does not work, because satisfying condition A requires that every loyal general obtain the same values $v(1), \ldots, v(n)$, and a traitorous general may send different values to different generals. For condition A to be satisfied, the following must be true:

1. Every loyal general must obtain the same information $v(1), \ldots, v(n)$.

Condition 1 implies that a general cannot necessarily use a value of $v(i)$ obtained directly from the ith general, since a traitorous ith general may send different values to different generals. This means that unless we are careful, in meeting condition 1 we might introduce the possibility that the generals use a value of $v(i)$ different from the one sent by the ith general—even though the ith general is loyal. We must not allow this to happen if condition B is to be met. For example, we cannot permit a few traitors to cause the loyal generals to base their decision upon the values "retreat", ..., "retreat" if every loyal general sent the value "attack". We therefore have the following requirement for each i:

2. If the ith general is loyal, then the value that he sends must be used by every loyal general as the value of $v(i)$.

We can rewrite condition 1 as the condition that for every i (whether or not the ith general is loyal),

1′. Any two loyal generals use the same value of $v(i)$.

Conditions 1′ and 2 are both conditions on the single value sent by the ith general. We can therefore restrict our consideration to the problem of how a single general sends his value to the others. We phrase this in terms of a commanding general sending an order to his lieutenants, obtaining the following problem.

Byzantine Generals Problem. A commanding general must send an order to his $n - 1$ lieutenant generals such that

IC1. All loyal lieutenants obey the same order.
IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

Conditions IC1 and IC2 are called the *interactive consistency* conditions. Note that if the commander is loyal, then IC1 follows from IC2. However, the commander need not be loyal.

To solve our original problem, the ith general sends his value of $v(i)$ by using a solution to the Byzantine Generals Problem to send the order "use $v(i)$ as my value", with the other generals acting as the lieutenants.

2. IMPOSSIBILITY RESULTS

The Byzantine Generals Problem seems deceptively simple. Its difficulty is indicated by the surprising fact that if the generals can send only oral messages, then no solution will work unless more than two-thirds of the generals are loyal. In particular, with only three generals, no solution can work in the presence of a single traitor. An oral message is one whose contents are completely under the control of the sender, so a traitorous sender can transmit any possible message. Such a message corresponds to the type of message that computers normally send to one another. In Section 4 we consider signed, written messages, for which this is not true.

We now show that with oral messages no solution for three generals can handle a single traitor. For simplicity, we consider the case in which the only possible decisions are "attack" or "retreat". Let us first examine the scenario pictured in Figure 1 in which the commander is loyal and sends an "attack" order, but Lieutenant 2 is a traitor and reports to Lieutenant 1 that he received a "retreat" order. For IC2 to be satisfied, Lieutenant 1 must obey the order to attack.

Now consider another scenario, shown in Figure 2, in which the commander is a traitor and sends an "attack" order to Lieutenant 1 and a "retreat" order to Lieutenant 2. Lieutenant 1 does not know who the traitor is, and he cannot tell what message the commander actually sent to Lieutenant 2. Hence, the scenarios in these two pictures appear exactly the same to Lieutenant 1. If the traitor lies consistently, then there is no way for Lieutenant 1 to distinguish between these two situations, so he must obey the "attack" order in both of them. Hence, whenever Lieutenant 1 receives an "attack" order from the commander, he must obey it.

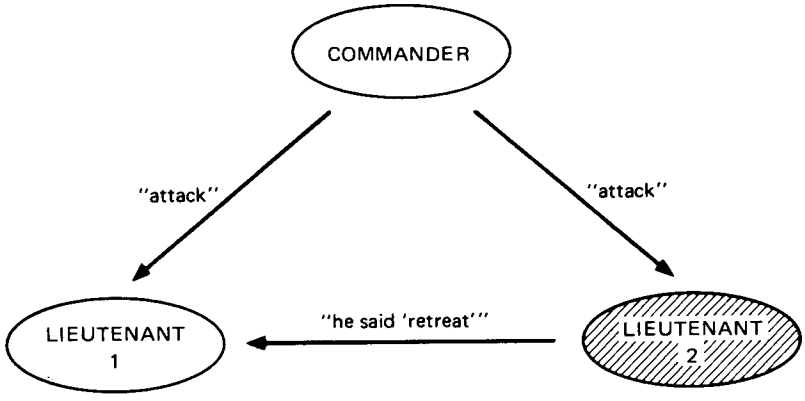

Fig. 1. Lieutenant 2 a traitor.

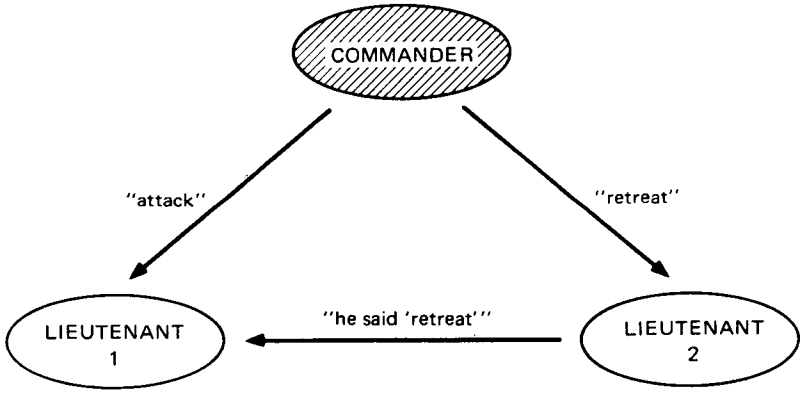

Fig. 2. The commander a traitor.

However, a similar argument shows that if Lieutenant 2 receives a "retreat" order from the commander then he must obey it even if Lieutenant 1 tells him that the commander said "attack". Therefore, in the scenario of Figure 2, Lieutenant 2 must obey the "retreat" order while Lieutenant 1 obeys the "attack" order, thereby violating condition IC1. Hence, no solution exists for three generals that works in the presence of a single traitor.

This argument may appear convincing, but we strongly advise the reader to be very suspicious of such nonrigorous reasoning. Although this result is indeed correct, we have seen equally plausible "proofs" of invalid results. We know of no area in computer science or mathematics in which informal reasoning is more likely to lead to errors than in the study of this type of algorithm. For a rigorous proof of the impossibility of a three-general solution that can handle a single traitor, we refer the reader to [3].

Using this result, we can show that no solution with fewer than $3m + 1$ generals can cope with m traitors.[1] The proof is by contradiction—we assume such a

[1] More precisely, no such solution exists for three or more generals, since the problem is trivial for two generals.

solution for a group of $3m$ or fewer and use it to construct a three-general solution to the Byzantine Generals Problem that works with one traitor, which we know to be impossible. To avoid confusion between the two algorithms, we call the generals of the assumed solution Albanian generals, and those of the constructed solution Byzantine generals. Thus, starting from an algorithm that allows $3m$ or fewer Albanian generals to cope with m traitors, we construct a solution that allows three Byzantine generals to handle a single traitor.

The three-general solution is obtained by having each of the Byzantine generals simulate approximately one-third of the Albanian generals, so that each Byzantine general is simulating at most m Albanian generals. The Byzantine commander simulates the Albanian commander plus at most $m - 1$ Albanian lieutenants, and each of the two Byzantine lieutenants simulates at most m Albanian lieutenants. Since only one Byzantine general can be a traitor, and he simulates at most m Albanians, at most m of the Albanian generals are traitors. Hence, the assumed solution guarantees that IC1 and IC2 hold for the Albanian generals. By IC1, all the Albanian lieutenants being simulated by a loyal Byzantine lieutenant obey the same order, which is the order he is to obey. It is easy to check that conditions IC1 and IC2 of the Albanian generals solution imply the corresponding conditions for the Byzantine generals, so we have constructed the required impossible solution.

One might think that the difficulty in solving the Byzantine Generals Problem stems from the requirement of reaching exact agreement. We now demonstrate that this is not the case by showing that reaching approximate agreement is just as hard as reaching exact agreement. Let us assume that instead of trying to agree on a precise battle plan, the generals must agree only upon an approximate time of attack. More precisely, we assume that the commander orders the time of the attack, and we require the following two conditions to hold:

IC1′. All loyal lieutenants attack within 10 minutes of one another.
IC2′. If the commanding general is loyal, then every loyal lieutenant attacks within 10 minutes of the time given in the commander's order.

(We assume that the orders are given and processed the day before the attack and that the time at which an order is received is irrelevant—only the attack time given in the order matters.)

Like the Byzantine Generals Problem, this problem is unsolvable unless more than two-thirds of the generals are loyal. We prove this by first showing that if there were a solution for three generals that coped with one traitor, then we could construct a three-general solution to the Byzantine Generals Problem that also worked in the presence of one traitor. Suppose the commander wishes to send an "attack" or "retreat" order. He orders an attack by sending an attack time of 1:00 and orders a retreat by sending an attack time of 2:00, using the assumed algorithm. Each lieutenant uses the following procedure to obtain his order.

(1) After receiving the attack time from the commander, a lieutenant does one of the following:

    (a) If the time is 1:10 or earlier, then attack.
    (b) If the time is 1:50 or later, then retreat.
    (c) Otherwise, continue to step (2).

(2) Ask the other lieutenant what decision he reached in step (1).

(a) If the other lieutenant reached a decision, then make the same decision he did.
(b) Otherwise, retreat.

It follows from IC2′ that if the commander is loyal, then a loyal lieutenant will obtain the correct order in step (1), so IC2 is satisfied. If the commander is loyal, then IC1 follows from IC2, so we need only prove IC1 under the assumption that the commander is a traitor. Since there is at most one traitor, this means that both lieutenants are loyal. It follows from IC1′ that if one lieutenant decides to attack in step (1), then the other cannot decide to retreat in step (1). Hence, either they will both come to the same decision in step (1) or at least one of them will defer his decision until step (2). In this case, it is easy to see that they both arrive at the same decision, so IC1 is satisfied. We have therefore constructed a three-general solution to the Byzantine Generals Problem that handles one traitor, which is impossible. Hence, we cannot have a three-general algorithm that maintains IC1′ and IC2′ in the presence of a traitor.

The method of having one general simulate m others can now be used to prove that no solution with fewer than $3m + 1$ generals can cope with m traitors. The proof is similar to the one for the original Byzantine Generals Problem and is left to the reader.

3. A SOLUTION WITH ORAL MESSAGES

We showed above that for a solution to the Byzantine Generals Problem using oral messages to cope with m traitors, there must be at least $3m + 1$ generals. We now give a solution that works for $3m + 1$ or more generals. However, we first specify exactly what we mean by "oral messages". Each general is supposed to execute some algorithm that involves sending messages to the other generals, and we assume that a loyal general correctly executes his algorithm. The definition of an oral message is embodied in the following assumptions which we make for the generals' message system:

A1. Every message that is sent is delivered correctly.
A2. The receiver of a message knows who sent it.
A3. The absence of a message can be detected.

Assumptions A1 and A2 prevent a traitor from interfering with the communication between two other generals, since by A1 he cannot interfere with the messages they do send, and by A2 he cannot confuse their intercourse by introducing spurious messages. Assumption A3 will foil a traitor who tries to prevent a decision by simply not sending messages. The practical implementation of these assumptions is discussed in Section 6.

The algorithms in this section and in the following one require that each general be able to send messages directly to every other general. In Section 5, we describe algorithms which do not have this requirement.

A traitorous commander may decide not to send any order. Since the lieutenants must obey some order, they need some default order to obey in this case. We let RETREAT be this default order.

We inductively define the *O*ral *M*essage algorithms OM(m), for all nonnegative integers m, by which a commander sends an order to $n - 1$ lieutenants. We show that OM(m) solves the Byzantine Generals Problem for $3m + 1$ or more generals in the presence of at most m traitors. We find it more convenient to describe this algorithm in terms of the lieutenants "obtaining a value" rather than "obeying an order".

The algorithm assumes a function *majority* with the property that if a majority of the values v_i equal v, then *majority*$(v_1, \ldots, v_{n-1})$ equals v. (Actually, it assumes a sequence of such functions—one for each n.) There are two natural choices for the value of *majority*$(v_1, \ldots, v_{n-1})$:

1. The majority value among the v_i if it exists, otherwise the value RETREAT;
2. The median of the v_i, assuming that they come from an ordered set.

The following algorithm requires only the aforementioned property of *majority*.

Algorithm OM(0).

(1) The commander sends his value to every lieutenant.
(2) Each lieutenant uses the value he receives from the commander, or uses the value RETREAT if he receives no value.

Algorithm OM(m), $m > 0$.

(1) The commander sends his value to every lieutenant.
(2) For each i, let v_i be the value Lieutenant i receives from the commander, or else be RETREAT if he receives no value. Lieutenant i acts as the commander in Algorithm OM($m - 1$) to send the value v_i to each of the $n - 2$ other lieutenants.
(3) For each i, and each $j \neq i$, let v_j be the value Lieutenant i received from Lieutenant j in step (2) (using Algorithm OM($m - 1$)), or else RETREAT if he received no such value. Lieutenant i uses the value *majority*$(v_1, \ldots, v_{n-1})$.

To understand how this algorithm works, we consider the case $m = 1$, $n = 4$. Figure 3 illustrates the messages received by Lieutenant 2 when the commander sends the value v and Lieutenant 3 is a traitor. In the first step of OM(1), the commander sends v to all three lieutenants. In the second step, Lieutenant 1 sends the value v to Lieutenant 2, using the trivial algorithm OM(0). Also in the second step, the traitorous Lieutenant 3 sends Lieutenant 2 some other value x. In step 3, Lieutenant 2 then has $v_1 = v_2 = v$ and $v_3 = x$, so he obtains the correct value $v = $ *majority*(v, v, x).

Next, we see what happens if the commander is a traitor. Figure 4 shows the values received by the lieutenants if a traitorous commander sends three arbitrary values x, y, and z to the three lieutenants. Each lieutenant obtains $v_1 = x$, $v_2 = y$, and $v_3 = z$, so they all obtain the same value *majority*(x, y, z) in step (3), regardless of whether or not any of the three values x, y, and z are equal.

The recursive algorithm OM(m) invokes $n - 1$ separate executions of the algorithm OM($m - 1$), each of which invokes $n - 2$ executions of OM($m - 2$), etc. This means that, for $m > 1$, a lieutenant sends many separate messages to each other lieutenant. There must be some way to distinguish among these different messages. The reader can verify that all ambiguity is removed if each lieutenant i prefixes the number i to the value v_i that he sends in step (2). As the recursion "unfolds," the algorithm OM($m - k$) will be called $(n - 1) \cdots (n - k)$ times to send a value prefixed by a sequence of k lieutenants' numbers.

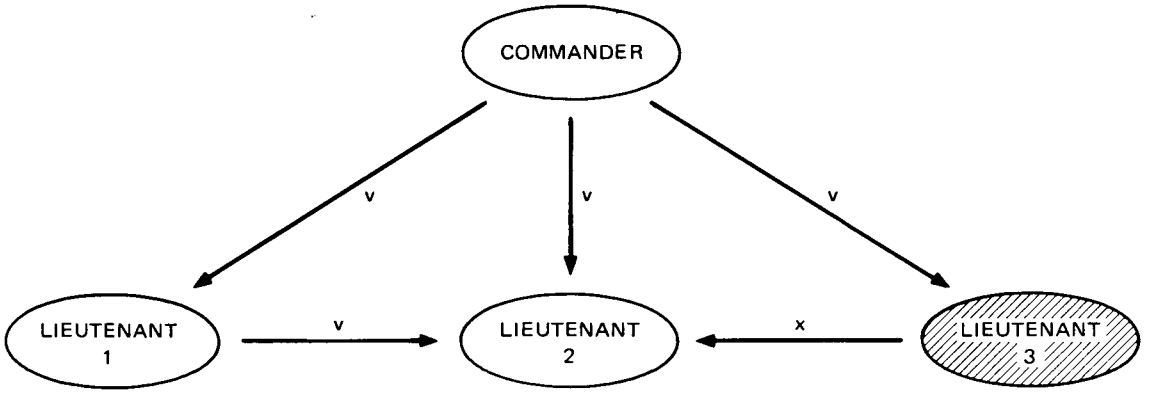

Fig. 3. Algorithm OM(1); Lieutenant 3 a traitor.

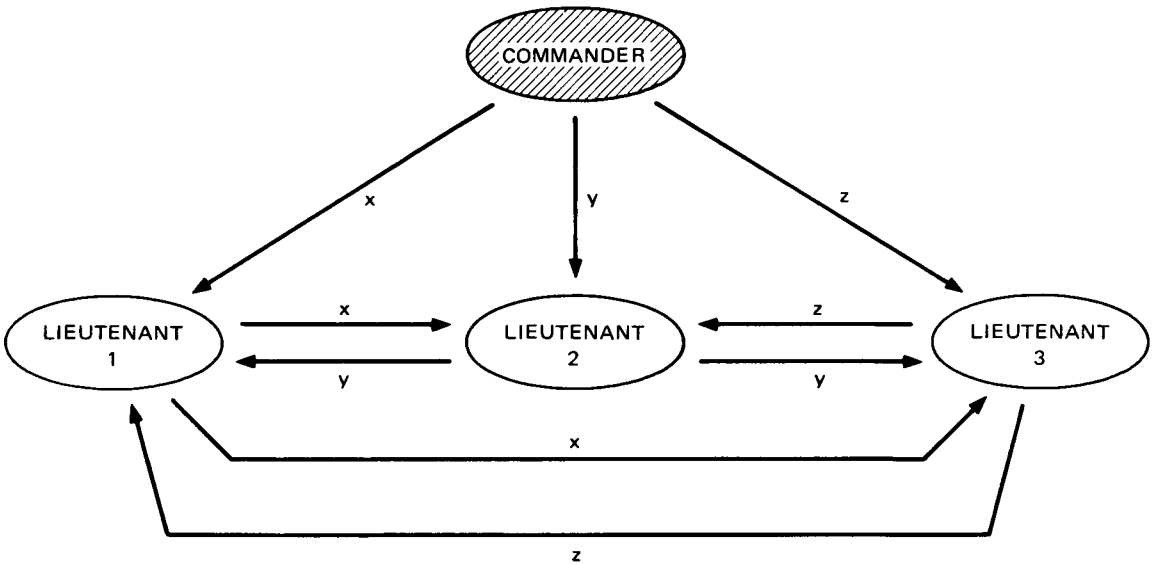

Fig. 4. Algorithm OM(1); the commander a traitor.

To prove the correctness of the algorithm OM(m) for arbitrary m, we first prove the following lemma.

LEMMA 1. *For any m and k, Algorithm OM(m) satisfies IC2 if there are more than $2k + m$ generals and at most k traitors.*

PROOF. The proof is by induction on m. IC2 only specifies what must happen if the commander is loyal. Using A1, it is easy to see that the trivial algorithm OM(0) works if the commander is loyal, so the lemma is true for $m = 0$. We now assume it is true for $m - 1$, $m > 0$, and prove it for m.

In step (1), the loyal commander sends a value v to all $n - 1$ lieutenants. In step (2), each loyal lieutenant applies OM($m - 1$) with $n - 1$ generals. Since by hypothesis $n > 2k + m$, we have $n - 1 > 2k + (m - 1)$, so we can apply the induction hypothesis to conclude that every loyal lieutenant gets $v_j = v$ for each loyal Lieutenant j. Since there are at most k traitors, and $n - 1 > 2k + (m - 1) \geq 2k$, a majority of the $n - 1$ lieutenants are loyal. Hence, each loyal lieutenant has $v_i = v$ for a majority of the $n - 1$ values i, so he obtains *majority*$(v_1, \ldots, v_{n-1}) = v$ in step (3), proving IC2. □

The following theorem asserts that Algorithm OM(m) solves the Byzantine Generals Problem.

THEOREM 1. *For any m, Algorithm OM(m) satisfies conditions IC1 and IC2 if there are more than $3m$ generals and at most m traitors.*

PROOF. The proof is by induction on m. If there are no traitors, then it is easy to see that OM(0) satisfies IC1 and IC2. We therefore assume that the theorem is true for OM($m - 1$) and prove it for OM(m), $m > 0$.

We first consider the case in which the commander is loyal. By taking k equal to m in Lemma 1, we see that OM(m) satisfies IC2. IC1 follows from IC2 if the commander is loyal, so we need only verify IC1 in the case that the commander is a traitor.

There are at most m traitors, and the commander is one of them, so at most $m - 1$ of the lieutenants are traitors. Since there are more than $3m$ generals, there are more than $3m - 1$ lieutenants, and $3m - 1 > 3(m - 1)$. We may therefore apply the induction hypothesis to conclude that OM($m - 1$) satisfies conditions IC1 and IC2. Hence, for each j, any two loyal lieutenants get the same value for v_j in step (3). (This follows from IC2 if one of the two lieutenants is Lieutenant j, and from IC1 otherwise.) Hence, any two loyal lieutenants get the same vector of values $v_1, \ldots, v_{n-1}$, and therefore obtain the same value *majority*$(v_1, \ldots, v_{n-1})$ in step (3), proving IC1. □

4. A SOLUTION WITH SIGNED MESSAGES

As we saw from the scenario of Figures 1 and 2, it is the traitors' ability to lie that makes the Byzantine Generals Problem so difficult. The problem becomes easier to solve if we can restrict that ability. One way to do this is to allow the generals to send unforgeable signed messages. More precisely, we add to A1–A3 the

following assumption:

A4 (a) A loyal general's signature cannot be forged, and any alteration of the contents of his signed messages can be detected.
(b) Anyone can verify the authenticity of a general's signature.

Note that we make no assumptions about a traitorous general's signature. In particular, we allow his signature to be forged by another traitor, thereby permitting collusion among the traitors.

Now that we have introduced signed messages, our previous argument that four generals are required to cope with one traitor no longer holds. In fact, a three-general solution does exist. We now give an algorithm that copes with m traitors for any number of generals. (The problem is vacuous if there are fewer than $m + 2$ generals.)

In our algorithm, the commander sends a signed order to each of his lieutenants. Each lieutenant then adds his signature to that order and sends it to the other lieutenants, who add their signatures and send it to others, and so on. This means that a lieutenant must effectively receive one signed message, make several copies of it, and sign and send those copies. It does not matter how these copies are obtained; a single message might be photocopied, or else each message might consist of a stack of identical messages which are signed and distributed as required.

Our algorithm assumes a function *choice* which is applied to a set of orders to obtain a single one. The only requirements we make for this function are

1. If the set V consists of the single element v, then $choice(V) = v$.
2. $choice(\varnothing)$ = RETREAT, where $\varnothing$ is the empty set.

Note that one possible definition is to let $choice(V)$ be the median element of V—assuming that there is an ordering of the elements.

In the following algorithm, we let $x:i$ denote the value x signed by General i. Thus, $v:j:i$ denotes the value v signed by j, and then that value $v:j$ signed by i. We let General 0 be the commander. In this algorithm, each lieutenant i maintains a set V_i, containng the set of properly signed orders he has received so far. (If the commander is loyal, then this set should never contain more than a single element.) Do not confuse V_i, the set of *orders* he has received, with the set of messages that he has received. There may be many different messages with the same order.

Algorithm SM(m).

Initially $V_i = \varnothing$.

(1) The commander signs and sends his value to every lieutenant.
(2) For each i:

(A) If Lieutenant i receives a message of the form $v:0$ from the commander and he has not yet received any order, then

(i) he lets V_i equal $\{v\}$;
(ii) he sends the message $v:0:i$ to every other lieutenant.

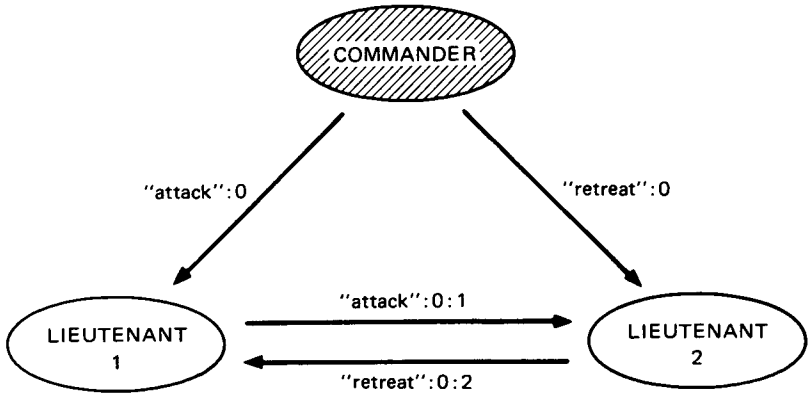

Fig. 5. Algorithm SM(1); the commander a traitor.

(B) If Lieutenant i receives a message of the form $v:0:j_1:\cdots:j_k$ and v is not in the set V_i, then
  (i) he adds v to V_i;
  (ii) if $k < m$, then he sends the message $v:0:j_1:\cdots:j_k:i$ to every lieutenant other than $j_1, \ldots, j_k$.

(3) For each i: When Lieutenant i will receive no more messages, he obeys the order $choice(V_i)$.

Note that in step (2), Lieutenant i ignores any message containing an order v that is already in the set V_i.

We have not specified how a lieutenant determines in step (3) that he will receive no more messages. By induction on k, one easily shows that for each sequence of lieutenants $j_1, \ldots, j_k$ with $k \leq m$, a lieutenant can receive at most one message of the form $v:0:j_1:\cdots:j_k$ in step (2). If we require that Lieutenant j_k either send such a message or else send a message reporting that he will not send such a message, then it is easy to decide when all messages have been received. (By assumption A3, a lieutenant can determine if a traitorous lieutenant j_k sends neither of those two messages.) Alternatively, time-out can be used to determine when no more messages will arrive. The use of time-out is discussed in Section 6.

Note that in step (2), Lieutenant i ignores any messages that do not have the proper form of a value followed by a string of signatures. If packets of identical messages are used to avoid having to copy messages, this means that he throws away any packet that does not consist of a sufficient number of identical, properly signed messages. (There should be $(n-k-2)(n-k-3)\cdots(n-m-2)$ copies of the message if it has been signed by k lieutenants.)

Figure 5 illustrates Algorithm SM(1) for the case of three generals when the commander is a traitor. The commander sends an "attack" order to one lieutenant and a "retreat" order to the other. Both lieutenants receive the two orders in step (2), so after step (2) $V_1 = V_2 = \{$"attack", "retreat"$\}$, and they both obey the order $choice(\{$"attack", "retreat"$\})$. Observe that here, unlike the situation in Figure 2, the lieutenants know the commander is a traitor because his signature

appears on two different orders, and A4 states that only he could have generated those signatures.

In Algorithm SM(m), a lieutenant signs his name to acknowledge his receipt of an order. If he is the mth lieutenant to add his signature to the order, then that signature is not relayed to anyone else by its recipient, so it is superfluous. (More precisely, assumption A2 makes it unnecessary.) In particular, the lieutenants need not sign their messages in SM(1).

We now prove the correctness of our algorithm.

THEOREM 2. *For any m, Algorithm SM(m) solves the Byzantine Generals Problem if there are at most m traitors.*

PROOF. We first prove IC2. If the commander is loyal, then he sends his signed order $v:0$ to every lieutenant in step (1). Every loyal lieutenant will therefore receive the order v in step (2)(A). Moreover, since no traitorous lieutenant can forge any other message of the form $v':0$, a loyal lieutenant can receive no additional order in step (2)(B). Hence, for each loyal Lieutenant i, the set V_i obtained in step (2) consists of the single order v, which he will obey in step (3) by property 1 of the *choice* function. This proves IC2.

Since IC1 follows from IC2 if the commander is loyal, to prove IC1 we need only consider the case in which the commander is a traitor. Two loyal lieutenants i and j obey the same order in step (3) if the sets of orders V_i and V_j that they receive in step (2) are the same. Therefore, to prove IC1 it suffices to prove that, if i puts an order v into V_i in step (2), then j must put the same order v into V_j in step (2). To do this, we must show that j receives a properly signed message containing that order. If i receives the order v in step (2)(A), then he sends it to j in step (2)(A)(ii); so j receives it (by A1). If i adds the order to V_i in step (2)(B), then he must receive a first message of the form $v:0:j_1:\cdots:j_k$. If j is one of the j_r, then by A4 he must already have received the order v. If not, we consider two cases:

1. $k < m$. In this case, i sends the message $v:0:j_1:\cdots:j_k:i$ to j; so j must receive the order v.

2. $k = m$. Since the commander is a traitor, at most $m - 1$ of the lieutenants are traitors. Hence, at least one of the lieutenants $j_1, \ldots, j_m$ is loyal. This loyal lieutenant must have sent j the value v when he first received it, so j must therefore receive that value.

This completes the proof. □

5. MISSING COMMUNICATION PATHS

Thus far, we have assumed that a general can send messages directly to every other general. We now remove this assumption. Instead, we suppose that physical barriers place some restrictions on who can send messages to whom. We consider the generals to form the nodes of a simple,[2] finite undirected graph G, where an arc between two nodes indicates that those two generals can send messages

[2] A simple graph is one in which there is at most one arc joining any two nodes, and every arc connects two distinct nodes.

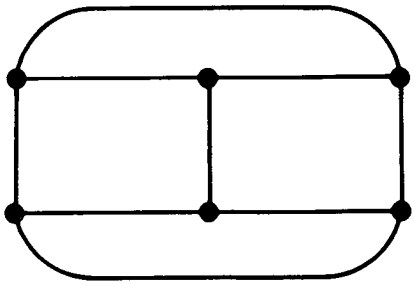

Fig. 6. A 3-regular graph.

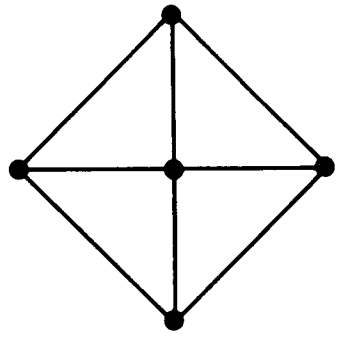

Fig. 7. A graph that is not 3-regular.

directly to one another. We now extend Algorithms OM(m) and SM(m), which assumed G to be completely connected, to more general graphs.

To extend our oral message algorithm OM(m), we need the following definition, where two generals are said to be *neighbors* if they are joined by an arc.

Definition 1.

(a) A set of nodes $\{i_1, \ldots, i_p\}$ is said to be a *regular set of neighbors* of a node i if

(i) each i_j is a neighbor of i, and
(ii) for any general k different from i, there exist paths $\gamma_{j,k}$ from i_j to k not passing through i such that any two different paths $\gamma_{j,k}$ have no node in common other than k.

(b) The graph G is said to be *p-regular* if every node has a regular set of neighbors consisting of p distinct nodes.

Figure 6 shows an example of a simple 3-regular graph. Figure 7 shows an example of a graph that is not 3-regular because the central node has no regular set of neighbors containing three nodes.

We extend OM(m) to an algorithm that solves the Byzantine Generals Problem in the presence of m traitors if the graph G of generals is $3m$-regular. (Note that a $3m$-regular graph must contain at least $3m + 1$ nodes.) For all *positive* integers m and p, we define the algorithm OM(m, p) as follows when the graph G of generals is p-regular. (OM(m, p) is not defined if G is not p-regular.) The definition uses induction on m.

Algorithm *OM*(m, p).

(0) Choose a regular set N of neighbors of the commander consisting of p lieutenants.
(1) The commander sends his value to every lieutenant in N.
(2) For each i in N, let v_i be the value Lieutenant i receives from the commander, or else RETREAT if he receives no value. Lieutenant i sends v_i to every other lieutenant k as follows:
    (A) If $m = 1$, then by sending the value along the path $\gamma_{i,k}$ whose existence is guaranteed by part (a)(ii) of Definition 1.
    (B) If $m > 1$, then by acting as the commander in the algorithm OM($m - 1, p - 1$), with the graph of generals obtained by removing the original commander from G.

(3) For each k, and each i in N with $i \neq k$, let v_i be the value Lieutenant k received from Lieutenant i in step (2), or RETREAT if he received no value. Lieutenant k uses the value $majority(v_{i_1}, \ldots, v_{i_p})$, where $N = \{i_1, \ldots, i_p\}$.

Note that removing a single node from a p-regular graph leaves a $(p - 1)$-regular graph. Hence, one can apply the algorithm OM$(m - 1, p - 1)$ in step (2)(B).

We now prove that OM$(m, 3m)$ solves the Byzantine Generals Problem if there are at most m traitors. The proof is similar to the proof for the algorithm OM(m) and will just be sketched. It begins with the following extension of Lemma 1.

LEMMA 2. *For any $m > 0$ and any $p \geq 2k + m$, Algorithm OM(m, p) satisfies IC2 if there are at most k traitors.*

PROOF. For $m = 1$, observe that a lieutenant obtains the value $majority(v_1, \ldots, v_p)$, where each v_i is a value sent to him by the commander along a path disjoint from the path used to send the other values to him. Since there are at most k traitors and $p \geq 2k + 1$, more than half of those paths are composed entirely of loyal lieutenants. Hence, if the commander is loyal, then a majority of the values v_i will equal the value he sent, which implies that IC2 is satisfied.

Now assume the lemma for $m - 1$, $m > 1$. If the commander is loyal, then each of the p lieutenants in N gets the correct value. Since $p > 2k$, a majority of them are loyal, and by the induction hypothesis each of them sends the correct value to every loyal lieutenant. Hence, each loyal lieutenant gets a majority of correct values, thereby obtaining the correct value in step (3). □

The correctness of Algorithm OM$(m, 3m)$ is an immediate consequence of the following result.

THEOREM 3. *For any $m > 0$ and any $p \geq 3m$, Algorithm OM(m, p) solves the Byzantine Generals Problem if there are at most m traitors.*

PROOF. By Lemma 2, letting $k = m$, we see that OM(m, p) satisfies IC2. If the commander is loyal, then IC1 follows from IC2, so we need only prove IC1 under the assumption that the commander is a traitor. To do this, we prove that every loyal lieutenant gets the same set of values v_i in step (3). If $m = 1$, then this follows because all the lieutenants, including those in N, are loyal and the paths $\gamma_{i,k}$ do not pass through the commander. For $m > 1$, a simple induction argument can be applied, since $p \geq 3m$ implies that $p - 1 \geq 3(m - 1)$. □

Our extension of Algorithm OM(m) requires that the graph G be $3m$-regular, which is a rather strong connectivity hypothesis.[3] In fact, if there are only $3m + 1$ generals (the minimum number required), then $3m$-regularity means complete connectivity, and Algorithm OM$(m, 3m)$ reduces to Algorithm OM(m). In contrast, Algorithm SM(m) is easily extended to allow the weakest possible connectivity hypothesis. Let us first consider how much connectivity is needed for the Byzantine Generals Problem to be solvable. IC2 requires that a loyal lieutenant obey a loyal commander. This is clearly impossible if the commander cannot communicate with the lieutenant. In particular, if every message from the

[3] A recent algorithm of Dolev [2] requires less connectivity.

commander to the lieutenant must be relayed by traitors, then there is no way to guarantee that the lieutenant gets the commander's order. Similarly, IC1 cannot be guaranteed if there are two lieutenants who can only communicate with one another via traitorous intermediaries.

The weakest connectivity hypothesis for which the Byzantine Generals Problem is solvable is that the subgraph formed by the loyal generals be connected. We show that under this hypothesis, the algorithm SM($n - 2$) is a solution, where n is the number of generals—regardless of the number of traitors. Of course, we must modify the algorithm so that generals only send messages to where they can be sent. More precisely, in step (1), the commander sends his signed order only to his neighboring lieutenants; and in step (2)(B), Lieutenant i only sends the message to every *neighboring* lieutenant not among the j_r.

We prove the following more general result, where the *diameter* of a graph is the smallest number d such that any two nodes are connected by a path containing at most d arcs.

THEOREM 4. *For any m and d, if there are at most m traitors and the subgraph of loyal generals has diameter d, then Algorithm SM($m + d - 1$) (with the above modification) solves the Byzantine Generals Problem.*

PROOF. The proof is quite similar to that of Theorem 2 and is just sketched here. To prove IC2, observe that by hypothesis there is a path from the loyal commander to a lieutenant i going through $d - 1$ or fewer loyal lieutenants. Those lieutenants will correctly relay the order until it reaches i. As before, assumption A4 prevents a traitor from forging a different order.

To prove IC1, we assume the commander is a traitor and must show that any order received by a loyal lieutenant i is also received by a loyal lieutenant j. Suppose i receives an order $v:0:j_1:\cdots:j_k$ not signed by j. If $k < m$, then i will send it to every neighbor who has not already received that order, and it will be relayed to j within $d - 1$ more steps. If $k \geq m$, then one of the first m signers must be loyal and must have sent it to all of his neighbors, whereupon it will be relayed by loyal generals and will reach j within $d - 1$ steps. □

COROLLARY. *If the graph of loyal generals is connected, then SM($n - 2$) (as modified above) solves the Byzantine Generals Problem for n generals.*

PROOF. Let d be the diameter of the graph of loyal generals. Since the diameter of a connected graph is less than the number of nodes, there must be more than d loyal generals and fewer than $n - d$ traitors. The result follows from the theorem by letting $m = n - d - 1$. □

Theorem 4 assumes that the subgraph of loyal generals is connected. Its proof is easily extended to show that even if this is not the case, if there are at most m traitors, then the algorithm SM($m + d - 1$) has the following two properties:

1. Any two loyal generals connected by a path of length at most d passing through only loyal generals will obey the same order.
2. If the commander is loyal, then any loyal lieutenant connected to him by a path of length at most $m + d$ passing only through loyal generals will obey his order.

6. RELIABLE SYSTEMS

Other than using intrinsically reliable circuit components, the only way we know to implement a reliable computer system is to use several different "processors" to compute the same result, and then to perform a majority vote on their outputs to obtain a single value. (The voting may be performed within the system, or externally by the users of the output.) This is true whether one is implementing a reliable computer using redundant circuitry to protect against the failure of individual chips, or a ballistic missile defense system using redundant computing sites to protect against the destruction of individual sites by a nuclear attack. The only difference is in the size of the replicated "processor".

The use of majority voting to achieve reliability is based upon the assumption that all the nonfaulty processors will produce the same output. This is true so long as they all use the same input. However, any single input datum comes from a single physical component—for example, from some other circuit in the reliable computer, or from some radar site in the missile defense system—and a malfunctioning component can give different values to different processors. Moreover, different processors can get different values even from a nonfaulty input unit if they read the value while it is changing. For example, if two processors read a clock while it is advancing, then one may get the old time and the other the new time. This can only be prevented by synchronizing the reads with the advancing of the clock.

In order for majority voting to yield a reliable system, the following two conditions should be satisfied:

1. All nonfaulty processors must use the same input value (so they produce the same output).
2. If the input unit is nonfaulty, then all nonfaulty processes use the value it provides as input (so they produce the correct output).

These are just our interactive consistency conditions IC1 and IC2, where the "commander" is the unit generating the input, the "lieutenants" are the processors, and "loyal" means nonfaulty.

It is tempting to try to circumvent the problem with a "hardware" solution. For example, one might try to insure that all processors obtain the same input value by having them all read it from the same wire. However, a faulty input unit could send a marginal signal along the wire—a signal that can be interpreted by some processors as a 0 and by others as a 1. There is no way to guarantee that different processors will get the same value from a possibly faulty input device except by having the processors communicate among themselves to solve the Byzantine Generals Problem.

Of course, a faulty input device may provide meaningless input values. All that a Byzantine Generals solution can do is guarantee that all processors use the same input value. If the input is an important one, then there should be several separate input devices providing redundant values. For example, there should be redundant radars as well as redundant processing sites in a missile defense system. However, redundant inputs cannot achieve reliability; it is still necessary to insure that the nonfaulty processors use the redundant data to produce the same output.

In case the input device is nonfaulty but gives different values because it is read while its value is changing, we still want the nonfaulty processors to obtain a reasonable input value. It can be shown that, if the functions *majority* and *choice* are taken to be the median functions, then our algorithms have the property that the value obtained by the nonfaulty processors lies within the range of values provided by the input unit. Thus, the nonfaulty processors will obtain a reasonable value so long as the input unit produces a reasonable range of values.

We have given several solutions, but they have been stated in terms of Byzantine generals rather than in terms of computing systems. We now examine how these solutions can be applied to reliable computing systems. Of course, there is no problem implementing a "general's" algorithm with a processor. The problems lie in implementing a message passing system that meets assumptions A1–A3 (assumptions A1–A4 for Algorithm SM(m)). We now consider these assumptions in order.

A1. Assumption A1 states that every message sent by a nonfaulty processor is delivered correctly. In real systems, communication lines can fail. For the oral message algorithms OM(m) and OM(m, p), the failure of the communication line joining two processors is indistinguishable from the failure of one of the processors. Hence, we can only guarantee that these algorithms will work in the presence of up to m failures, be they processor or communication line failures. (Of course, the failure of several communication lines attached to the same processor is equivalent to a single processor failure.) If we assume that a failed communication line cannot result in the forgery of a signed message—an assumption which we will see below is quite reasonable, then our signed message algorithm SM(m) is insensitive to communication line failure. More precisely, Theorem 4 remains valid even with communication line failure. A failed communication line has the same effect as simply removing the communication line—it lowers the connectivity of the processor graph.

A2. Assumption A2 states that a processor can determine the originator of any message that it received. What is actually necessary is that a faulty processor not be able to impersonate a nonfaulty one. In practice, this means that interprocess communication be over fixed lines rather than through some message switching network. (If a switching network is used, then faulty network nodes must be considered, and the Byzantine Generals Problem appears again.) Note that assumption A2 is not needed if A4 is assumed and all messages are signed, since impersonation of another processor would imply forging its messages.

A3. Assumption A3 requires that the absence of a message can be detected. The absence of a message can only be detected by its failure to arrive within some fixed length of time—in other words, by the use of some time-out convention. The use of time-out to satisfy A3 requires two assumptions:

1. There is a fixed maximum time needed for the generation and transmission of a message.
2. The sender and receiver have clocks that are synchronized to within some fixed maximum error.

The need for the first assumption is fairly obvious, since the receiver must know

how long he needs to wait for the message to arrive. (The generation time is how long it takes the processor to send the message after receiving all the input necessary to generate it.) The need for the second assumption is less obvious. However, it can be shown that either this assumption or an equivalent one is necessary to solve the Byzantine Generals Problem. More precisely, suppose that we allow algorithms in which the generals take action only in the following circumstances:

1. At some fixed initial time (the same for all generals).
2. Upon the receipt of a message.
3. When a randomly chosen length of time has elapsed. (I.e., a general can set a timer to a random value and act when the timer goes off.)

(This yields the most general class of algorithms we can envision which does not allow the construction of synchronized clocks.) It can be shown that no such algorithm can solve the Byzantine Generals Problem if messages can be transmitted arbitrarily quickly, even if there is an upper bound on message transmission delay. Moreover, no solution is possible even if we restrict the traitors so that the only incorrect behavior they are permitted is the failure to send a message. The proof of this result is beyond the scope of this paper. Note that placing a lower as well as an upper bound on transmission delay allows processors to implement clocks by sending messages back and forth.

The above two assumptions make it easy to detect unsent messages. Let μ be the maximum message generation and transmission delay, and assume the nonfaulty processors have clocks that differ from one another by at most τ at any time. Then any message that a nonfaulty process should begin to generate by time T on its clock will arrive at its destination by time $T + \mu + \tau$ on the receiver's clock. Hence, if the receiver has not received the message by that time, then it may assume that it was not sent. (If it arrives later, then the sender must be faulty, so the correctness of our algorithms does not depend upon the message being sent.) By fixing the time at which the input processor sends its value, one can calculate until what time on its own clock a processor must wait for each message. For example, in Algorithm SM(m) a processor must wait until time $T_0 + k(\mu + \tau)$ for any message having k signatures, where T_0 is the time (on his clock) at which the commander starts executing the algorithm.

No two clocks run at precisely the same rate, so no matter how accurately the processors' clocks are synchronized initially, they will eventually drift arbitrarily far apart unless they are periodically resynchronized. We therefore have the problem of keeping the processors' clocks all synchronized to within some fixed amount, even if some of the processors are faulty. This is as difficult a problem as the Byzantine Generals Problem itself. Solutions to the clock synchronization problem exist which are closely related to our Byzantine Generals solutions. They will be described in a future paper.

A4. Assumption A4 requires that processors be able to sign their messages in such a way that a nonfaulty processor's signature cannot be forged. A signature is a piece of redundant information $S_i(M)$ generated by process i from a data item M. A message signed by i consists of a pair $(M, S_i(M))$. To meet parts (a)

and (b) of A4, the function S_i must have the following two properties:

(a) If processor i is nonfaulty, then no faulty processor can generate $S_i(M)$.
(b) Given M and X, any process can determine if X equals $S_i(M)$.

Property (a) can never be guaranteed, since $S_i(M)$ is just a data item, and a faulty processor could generate any data item. However, we can make the probability of its violation as small as we wish, thereby making the system as reliable as we wish. How this is done depends upon the type of faults we expect to encounter. There are two cases of interest:

1. *Random Malfunction.* By making S_i a suitably "randomizing" function, we can make the probability that a random malfunction in a processor generates a correct signature essentially equal to the probability of its doing so through a random choice procedure—that is, the reciprocal of the number of possible signatures. The following is one method for doing this. Assume that messages are encoded as positive integers less than P, where P is a power of two. Let $S_i(M)$ equal $M * K_i \bmod P$, where K_i is a randomly chosen odd number less than P. Letting K_i^{-1} be the unique number less than P such that $K_i * K_i^{-1} \equiv 1 \bmod P$, a process can check that $X = S_i(M)$ by testing that $M \equiv X * K_i^{-1} \bmod P$. If another processor does not have K_i in its memory, then the probability of its generating the correct signature $M * K_i$ for a single (nonzero) message M should be $1/P$: its probability of doing so by random choice. (Note that if the processor could obtain K_i by some simple procedure, then there might be a larger probability of a faulty processor j forging i's signature by substituting K_i for K_j when trying to compute $S_j(M)$.)

2. *Malicious Intelligence.* If the faulty processor is being guided by a malicious intelligence—for example, if it is a perfectly good processor being operated by a human who is trying to disrupt the system—then the construction of the signature function S_i becomes a cryptography problem. We refer the reader to [1] and [4] for a discussion of how this problem can be solved.

Note that it is easy to generate the signature $S_i(M)$ if the process has already seen that signature. Hence, it is important that the same message never have to be signed twice. This means that, when using SM(m) repeatedly to distribute a sequence of values, sequence numbers should be appended to the values to guarantee uniqueness.

7. CONCLUSION

We have presented several solutions to the Byzantine Generals Problem, under various hypotheses, and shown how they can be used in implementing reliable computer systems. These solutions are expensive in both the amount of time and the number of messages required. Algorithms OM(m) and SM(m) both require message paths of length up to $m + 1$. In other words, each lieutenant may have to wait for messages that originated at the commander and were then relayed via m other lieutenants. Fischer and Lynch have shown that this must be true for any solution that can cope with m traitors, so our solutions are optimal in that respect. Our algorithms for a graph that is not completely connected require

message paths of length up to $m + d$, where d is the diameter of the subgraph of loyal generals. We suspect that this is also optimal.

Algorithms OM(m) and SM(m) involve sending up to $(n - 1)(n - 2) \cdots (n - m - 1)$ messages. The number of separate messages required can certainly be reduced by combining messages. It may also be possible to reduce the amount of information transferred, but this has not been studied in detail. However, we expect that a large number of messages will still be required.

Achieving reliability in the face of arbitrary malfunctioning is a difficult problem, and its solution seems to be inherently expensive. The only way to reduce the cost is to make assumptions about the type of failure that may occur. For example, it is often assumed that a computer may fail to respond but will never respond incorrectly. However, when extremely high reliability is required, such assumptions cannot be made, and the full expense of a Byzantine Generals solution is required.

REFERENCES

1. DIFFIE, W., AND HELLMAN, M.E. New directions in cryptography. *IEEE Trans. Inf. Theory IT-22* (Nov. 1976), 644–654.
2. DOLEV, D. The Byzantine generals strike again. *J. Algorithms 3*, 1 (Jan. 1982).
3. PEASE, M., SHOSTAK, R., AND LAMPORT, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (Apr. 1980), 228–234.
4. RIVEST, R.L., SHAMIR, A., AND ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*, 2 (Feb. 1978), 120–126.

Received April 1980; revised November 1981; accepted November 1981