TOWARD A TOPOLOGICAL CHARACTERIZATION OF ASYNCHRONOUS COMPLEXITY*

GUNNAR HOEST† AND NIR SHAVIT‡

Abstract. This paper introduces the use of topological models and methods, formerly used to analyze computability, as tools for the quantification and classification of *asynchronous complexity*. We present the first *asynchronous complexity theorem*, applied to decision tasks in the iterated immediate snapshot (IIS) model of Borowsky and Gafni. We do so by introducing a novel form of topological tool called the *nonuniform chromatic subdivision*. Building on the framework of Herlihy and Shavit's topological computability model, our theorem states that the time complexity of any asynchronous algorithm 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. To show the power of our theorem, we use it to derive a new tight bound on the time to achieve n process approximate agreement in the IIS model: $\left\lceil \log_d \frac{\max\_input - \min\_input}{\epsilon} \right\rceil$, where $d = 3$ for two processes and $d = 2$ for three or more. This closes an intriguing gap between the known upper and lower bounds implied by the work of Aspnes and Herlihy. More than the new bounds themselves, the importance of our asynchronous complexity theorem is that the algorithms and lower bounds it allows us to derive are intuitive and simple, with topological proofs that require no mention of concurrency at all.

Key words. shared memory, asynchronous systems, topology, immediate snapshots, approximate agreement, simplicial complexes, subdivisions

AMS subject classifications. 68Q10, 68Q17, 68Q60, 68Q85

DOI. 10.1137/S0097539701397412

1. Introduction. In the 21st century, computers are progressively being used as coordination devices in asynchronous, distributed systems. Unfortunately, the standard, Turing notions of computability and complexity are not sufficient for evaluating the behavior of such systems. In the last few years, techniques of modeling and analysis based on classical algebraic topology [3, 12, 14, 18, 20, 21, 22, 24, 25, 26, 32] in conjunction with distributed simulation methods [9, 10, 11, 12] have brought about significant progress in our understanding of computability problems in an asynchronous distributed setting. We feel the time is ripe to extend these techniques to address *asynchronous complexity*.

This paper studies asynchronous shared memory solutions to the class of problems called *decision tasks*, input/output problems in which N processes start with input values and, after communicating, halt with private output values. We focus on the *iterated immediate snapshot* (IIS) memory model introduced by Borowsky and Gafni [12] as part of their new simplified proof of the asynchronous computability theorem [26]. The model is a restriction of atomic snapshot memory that guarantees that processes' scan operations return views that contain nondecreasing sets of the

participating processes' inputs. Though it is not a realistic computation model (no machine supports such operations) we believe it is a good first candidate for topological modeling since it has a particularly nice geometric representation and hence easily lends itself to topological analysis.

1.1. Historical background and related work. Let us begin by giving a brief account of previous work on computability problems in fault-prone, asynchronous, distributed systems, applications of algebraic topology to asynchronous computability problems, simulation techniques, and also on characterizing the *Approximate Agreement* task.

In 1985, a fundamental paper by Fischer, Lynch, and Paterson [17] demonstrated that traditional Turing computability theory is not sufficient for analyzing computability problems in asynchronous, distributed systems. In particular, it showed that the well-known *Consensus* task, in which each participating process has a private input value drawn from some set S, and every nonfaulty process must decide on the same output value equal to the input of some process, cannot be solved in a message passing system even if only one process may fail by halting. Later, it was also shown that the message passing and shared memory models are equivalent [4], so this result carries over to shared memory systems as well. This fundamental discovery led to the creation of a highly active research area, which is surveyed in a recent book by Lynch [29].

In 1988, Biran, Moran, and Zaks [7] provided a breakthrough result by introducing a graph-theoretic framework that allows a complete characterization of the types of tasks that can be solved in a message passing or shared memory system in the presence of a single failure. However, this framework proved hard to extend to more than one failure, and even the problem of characterizing the solvability of specific tasks such as *Renaming* [5] and *Set Agreement* [13] for any number of processes remains unsolved.

In 1993, three research teams working independently—Borowsky and Gafni [10], Saks and Zaharoglou [32], and Herlihy and Shavit [24]—derived impossibility results for solving the Set Agreement task in the read-write shared memory model. The paper of Borowsky and Gafni introduced a powerful new simulation technique for proving solvability and unsolvability results in asynchronous, distributed systems. The technique allows N-process protocols to be executed by fewer processes in a resilient way and has been proven correct by Borowsky, Gafni, Lynch, and Rajsbaum [9]. The paper by Saks and Zaharoglou [32] constructed an elegant topological structure that captures the knowledge of the processors of the state of the system, allowing them to prove the impossibility of wait-free k-set agreement using point-set topology. The proof exposes an interesting relation between set agreement and the Brouwer fixed point theorem for the k-dimensional ball.

The paper of Herlihy and Shavit [24, 26] introduced a new formalism based on tools from classical, algebraic topology for reasoning about computations in asynchronous, distributed systems in which any number of processes may fail. Their framework consisted of modeling tasks and protocols using algebraic structures called *simplicial complexes* and then applying standard homology theory to reason about them. Herlihy and Shavit extended this framework by providing the *asynchronous computability theorem*, which states a condition that is necessary and sufficient for a task to be solvable by a wait-free protocol in shared memory [26], and showed applications of this theorem to tasks such as Set Agreement and Renaming. Borowsky [8] generalized this solvability condition to a model consisting of regular shared memory augmented with set-consensus objects, under more general resiliency requirements.

In 1993, Chaudhuri, Herlihy, Lynch, and Tuttle [14] also used topological and geometric arguments to prove tight bounds on solving the Set Agreement problem in the *synchronous* message passing model where an arbitrary number of processes may fail.

In 1994, Herlihy and Rajsbaum derived further impossibility results for Set Agreement by applying classical homology theory [20]. Moreover, in a unifying paper in 1995, Herlihy and Rajsbaum provided a common, general framework for describing a wide collection of impossibility results by using chain maps and chain complexes [22]. At the same time, Attiya and Rajsbaum reproved several impossibility results using purely combinatorial tools [3].

In 1995, Gafni and Koutsopias presented a reduction from the classical contractibility problem of algebraic topology to show that it is undecidable whether a certain class of 3-process tasks is wait-free solvable in the shared memory model [18]. This work was then generalized by Herlihy and Rajsbaum to arbitrary numbers of processes and failures in a variety of computational models [21]. More recently, Havlicek showed that, while undecidability holds in the general case, the problem of solvability is in fact decidable for a relatively large class of tasks [19]. Another recent paper by Herlihy, Rajsbaum, and Tuttle [23] introduces the use of pseudospheres as a means for unifying the synchronous, semisynchronous, and asynchronous message passing computation models.

The immediate snapshot (IS) object was introduced by Borowsky and Gafni in 1993 [11]. It is the basic building block of the iterated immediate snapshot (IIS) model, first implicitly used by Herlihy and Shavit [24, 25] and more recently formulated as a computation model by Borowsky and Gafni [12] as part of their new, simplified proof of the asynchronous computability theorem of Herlihy and Shavit [26]. This work also shows that the IIS model is computationally equivalent to standard shared memory models by providing a wait-free implementation of IIS from shared memory, and vice versa. It is not clear, however, whether these implementations are optimal from a complexity-theoretic viewpoint.

The *Approximate Agreement* problem is a weakening of the *Consensus* problem in which each process has a real valued input, and in any execution, nonfaulty processes with inputs in a range [min_input, max_input] (the range changes from one execution to the next based on the input set of participating processes) must agree on output values within that range that are at most $\epsilon > 0$ apart. The problem was first introduced in 1986 by Dolev, Lynch, Pinter, Stark, and Weihl [15] in a paper showing that this task can be solved in both the synchronous and asynchronous message passing models even when assuming a Byzantine failure model (in which processes may exhibit arbitrary, even malicious, behavior). The paper also provided matching upper and lower bounds for solving the task in these settings. These results were extended to various failure models by Fekete [16], who also showed optimality in terms of the number of rounds of communication used.

In 1994, Attiya, Lynch, and Shavit published a paper giving an $\Omega(\log N)$ step complexity lower bound, together with a matching $O(\log N)$ upper bound, for solving in a wait-free manner N-process Approximate Agreement in "normal" (synchronous and failure-free) executions using single-writer, multireader shared memory [6]. These results were part of a proof that, in certain settings, wait-free algorithms are inherently *slower* than non–wait-free algorithms.

This work was extended by Schenk [33], who showed matching upper and lower bounds for solving the task in the asynchronous single-writer, multireader shared memory model where the magnitudes of the inputs are bounded from above.

Finally, in 1994, Aspnes and Herlihy [2] showed a $\left\lceil \log_3 \frac{\max\_input - \min\_input}{\epsilon} \right\rceil$ lower bound, together with a $\left\lceil \log_2 \frac{\max\_input - \min\_input}{\epsilon} \right\rceil$ upper bound on the time complexity (the number of steps taken by a process) for solving Approximate Agreement using wait-free protocols in the asynchronous single-writer, multireader shared memory model.

1.2. The asynchronous complexity theorem. This paper introduces a new theorem that for the first time provides a topological characterization of complexity for asynchronous computation. We introduce the *nonuniform iterated immediate snapshot (NIIS) model*, a refinement of the IIS model that allows better modeling of complexity. Keeping in style with Herlihy and Shavit's topological computability framework [26], our theorem states that the worst case time complexity for solving a decision task in the NIIS model is equivalent to the minimal number of nonuniform chromatic subdivisions of the task's input complex necessary to allow a simplicial map from the subdivided input complex to the output complex. The theorem implies an algorithm if one is given a subdivision and a mapping.

The nonuniform chromatic subdivisions we introduce (see Figure 13) are a looser and more general form of standard chromatic subdivisions [26]. Unlike the iterated standard chromatic subdivisions used in the computability work of [26, 12], they allow individual simplexes in a complex to be subdivided a different number of times, while ensuring that the subdivision of the complex as a whole remains consistent. Nonuniformity is a necessary property when analyzing complexity since it allows the level of subdivision of input simplexes to differ from one simplex to the next. This allows one to model a world in which different numbers of steps are taken on different input sets. If one used only uniform subdivisions, one could talk only about the complexity of the most highly subdivided simplex. This would make the complexity theorem useless, since, for example, for the Approximate Agreement problem, Aspnes and Herlihy [2] show that for any k one can find a set of inputs that will require time k in the worst case.

The power of our theorem lies in its ability to allow one to reason about the complexity of problems in a purely geometric setting. As we show, the subdivisions of a complex are a clean and higher level way of thinking about the multitude of different length executions of a concurrent protocol. We found this geometric representation helpful and believe that it will prove to be an invaluable tool for designing and analyzing concurrent algorithms. For technical reasons, in order to avoid the need to deal with infinite size complexes, we restrict our problem space to decision tasks with finite (yet not necessarily bounded) input and output domains.

We provide an example application of Theorem 4.2. In section 5, we use our topological framework to show tight upper and lower bounds on the time to solve the Approximate Agreement problem in a wait-free manner in the NIIS model. We close the gap implied by the work of Aspnes and Herlihy [2], proving matching upper and lower bounds of $\left\lceil \log_d \frac{\max\_input - \min\_input}{\epsilon} \right\rceil$, where $d = 3$ for two processes and $d = 2$ for three or more.

Apart from the theorem itself, its proof provides two additional contributions to the asynchronous computability literature.

- The upper bound proof of Herlihy and Shavit's asynchronous computability theorem [26] and related papers by Borowsky and Gafni [11, 12] all rely on the fact that the standard chromatic subdivision [25, 26] is indeed a subdivision in the topological sense. We provide the first formal proof of this fact.
- In 1997, Borowsky and Gafni provided a simulation of atomic snapshot mem-

ory from IIS memory [12]. They showed that based on this simulation, if one is given a proof of an asynchronous computability theorem for the IIS model (which they called Proposition 3.1), it will imply one for the general read-write model. The hope was that the proof of their Proposition 3.1 would be constructive and therefore significantly simpler than the nonconstructive proof in [24, 26]. The proof of our asynchronous complexity theorem in section 4 provides a constructive proof of computability for the NIIS model, and since IIS is a subset of NIIS, it provides the first known proof of Proposition 3.1 of [12].

1.3. Organization. The paper is organized as follows. Section 2 provides a formal definition of decision tasks. It also contains a thorough description of our model of computation, together with the complexity measures we use for analyzing protocols in this model. Section 3 contains a collection of necessary definitions and results from algebraic topology, as well as a description of how we model decision tasks and NIIS protocols topologically. It also contains definitions of the standard chromatic subdivision and the nonuniform chromatic subdivision. Section 4 contains a statement and proof of our main theorem. Section 5 contains an application of our asynchronous complexity theorem to the Approximate Agreement task. Finally, section 6 summarizes our results and also gives some directions for further research.

2. Model. In order to develop a useful and applicable complexity theory for asynchronous, distributed computer systems, we need to define some reasonable model of such systems. This model must be detailed enough so as to accurately and faithfully capture the inherent complexity of solving tasks in real distributed systems yet be simple enough so as to easily lend itself to some practical form of complexity analysis. The model we consider in this paper consists of a class of one-shot distributed problems, called *decision tasks*, together with a novel model of computation, a type of shared memory called the *nonuniform iterated immediate snapshot (NIIS) model*. This section contains a detailed description of these fundamental concepts. It also contains the complexity measures that will be used to analyze the complexity of solving decision tasks in the NIIS model.

2.1. Informal synopsis. We begin with an informal synopsis of our model, which largely follows that of Herlihy and Shavit [24, 25, 26]. Some fixed number $N = n+1$ of sequential threads of control, called *processes*, communicate by asynchronously accessing shared memory in order to solve *decision tasks*. In such a 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 [17]. A *protocol* is a distributed program that solves a decision task in such a system. A protocol is *wait-free* if it guarantees that every nonfaulty process will halt in a finite number of steps, independent of the progress of the other processes. The *time complexity* of solving a decision task in this model on a given input set is the supremum of the number of accesses to shared memory made by any process on that input set.

2.2. Decision tasks. In this section, we define decision tasks more precisely. This class of tasks is intended to provide a simplified model of reactive systems, such as databases, file systems, or automated teller machines. An input value represents information entering the system from the surrounding environment, such as a character typed at a keyboard, a message from another computer, or a signal from a sensor. An output value models an effect on the outside world, such as an irrevocable

decision to commit a transaction, to dispense cash, or to launch a missile. Informally speaking, a decision task is a relation between vectors of input values and vectors of output values. We define this more precisely below.

Let D_I and D_O be two finite data types, possibly identical, called the *input data type* and the *output data type*, respectively. We first define the concept of an input vector.

DEFINITION 2.1. *An $n+1$-process* input vector $\vec{I}$ *is an $n+1$-dimensional vector, indexed by $\{0, \ldots, n\}$, each component of which is either an object of type D_I or the distinguished value $\perp$, with the additional requirement that at least one component of $\vec{I}$ must be different from $\perp$.*

The definition of output vectors is similar to that of input vectors.

DEFINITION 2.2. *An $n+1$-process* output vector $\vec{O}$ *is an $n+1$-dimensional vector, indexed by $\{0, \ldots, n\}$, each component of which is either an object of type D_O or the distinguished value $\perp$.*

When it is clear from the context, we omit mentioning the number of processes in specifying input and output vectors. We denote the ith component of an input vector $\vec{I}$ by $\vec{I}[i]$, and, similarly, we denote the ith component of an output vector $\vec{O}$ by $\vec{O}[i]$. In the remainder of the paper, unless stated otherwise, we will assume that i and j are index values in the set $\{0, \ldots, n\}$. These index values will be used both for specifying vector elements and also for indexing processes. We will use the terms "one-dimensional array" ("array" for short) and "vector" interchangeably.

We are often concerned with executions that are prefixes of a given execution.

DEFINITION 2.3. *Vector $\vec{U}$ is a* prefix *of $\vec{V}$ if, for $0 \le i \le n$, either $\vec{U}[i] = \vec{V}[i]$ or $\vec{U}[i] = \perp$.*

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

DEFINITION 2.4. *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.*

In this paper, we will consider only sets of input and output vectors that are finite and prefix-closed.

DEFINITION 2.5. *An* input set *is a finite, prefix-closed set of input vectors. An* output set *is a finite, prefix-closed set of output vectors.*

Next, we define the notion of a task specification map, which maps each element of the input set to a subset of the output set. Our definition is similar to that of Havlicek [19].

DEFINITION 2.6. *Let I and O be input and output sets, respectively. A* task specification map *relating the two sets is a relation $\gamma \subseteq I \times O$ such that the following conditions hold:*

- *For all $\vec{I} \in I$, there exists a vector $\vec{O} \in O$ such that $(\vec{I}, \vec{O}) \in \gamma$.*
- *For all $(\vec{I}, \vec{O}) \in \gamma$, and for all i, $\vec{I}[i] = \perp$ if and only if $\vec{O}[i] = \perp$.*

Note that the above definition requires that in defining the task, every participating process must have a defined output. This is the specification of the task and does not model its solvability in any given computation model.

As a convenient notation, we denote the set of vectors $\vec{O}$ in O such that $(\vec{I}, \vec{O}) \in \gamma$ by $\gamma(\vec{I})$. For a given input vector $\vec{I}$, the set of vectors $\gamma(\vec{I})$ simply represents the set of legitimate output vectors for the set of inputs specified by $\vec{I}$. This set will generally contain more than one allowable output vector.

DEFINITION 2.7. *A decision task $\mathcal{D} = \langle I, O, \gamma \rangle$ is a tuple consisting of a set I of input vectors, a set O of output vectors, and a task specification map γ relating these two sets.*

We note that, by definition, decision tasks are inherently *one-shot* in the sense that all processes have a single input and must decide on a single output exactly once. Not all entries in a given input vector need contain an input value; some may contain the special value $\perp$, indicating that some processes do not receive an input value. We formalize this notion of participation in the definition below.

DEFINITION 2.8. *For any input vector $\vec{I}$, if the ith component is not $\perp$, then i participates in $\vec{I}$. Otherwise, we say that i does not participate in $\vec{I}$. Moreover, we define the participating set in $\vec{I}$ to be the set of participating indexes.*

As noted by Herlihy and Shavit [24, 25, 26], the reason for incorporating an explicit notion of participating indexes in our formalism for decision tasks is that it is convenient for capturing the intuitive notion of "order of actions in time" through the use of participating processes. For example, it allows us to distinguish between tasks such as *Unique-Id* and *Fetch-And-Increment*, which have the same sets of input and output vectors, have the same $\gamma(\mathcal{I})$ when all processes participate, but have quite different task specification maps when subsets of participating processes are taken into account.

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

FIG. 1. *The Unique-Id task.*

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

FIG. 2. *The Fetch-And-Increment task.*

EXAMPLE 2.9. *The $n + 1$-process* Unique-Id *task is defined as follows: each participating process $i \in \{0, \ldots, n\}$ has an input $x_i = 0$ and chooses an output $y_i \in \{0, \ldots, n\}$ such that for any pair of processes $i \neq j$, $y_i \neq y_j$.*

EXAMPLE 2.10. *In the* Fetch-And-Increment *problem, each participating process $i \in \{0, \ldots, n\}$ has an input $x_i = 0$ and chooses a unique output $y_i \in \{0, \ldots, n\}$ such that (1) for some participating process 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 Figures 1 and 2, taken from [26], show the task specifications for *Unique-Id* and *Fetch-And-Increment* for three processes. Notice that *Unique-Id* allows identifiers to be assigned statically, while *Fetch-And-Increment* effectively requires

that they be assigned dynamically in increasing order. The first task has a trivial wait-free solution: statically preassign the values 0, 1, and 2 to the three processes. The second has no solution in read-write memory if one or more processes can fail.

2.3. Modeling objects, processes, and protocols. We formally model objects, processes, and protocols using a simplified form of the Input/Output (I/O) automaton formalism of Lynch and Tuttle [28]. An *I/O automaton* is a nondeterministic automaton with a finite or infinite set of *states*, a set of *input actions*, a set of *output actions*, and a transition relation given by a set of *steps*, each defining a state transition following a given action. An *execution* of an I/O automaton is an alternating sequence of states and enabled actions, starting from some initial state. An *execution fragment* is a subsequence of consecutive states and actions 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 the context. An automaton *history* is the subsequence of actions occurring in an execution. Automata can be composed by identifying input and output actions in the natural way (details can be found in [28]).

An *object X* is an automaton with input action $call(i, v, X, D)$ and output action $return(i, v, X, D)$, where i is a process id, v is a value, X an object, and D a data type. An action on object X by process i is said to occur on X's ith "port." A *process i* is an automaton with output actions $call(i, v, X, D)$, and $decide(i, v)$ and input actions $return(i, v, X, D)$ and $start(i, v)$. An *operation* is a *matching* pair of *call* and *return* actions, that is, having the same type, name, and process id. From here on we will abuse this notation for the sake of clarity by dropping unnecessary parameters and denoting others using subscripts.

A *protocol* $\mathcal{P} = \{0, \ldots, n; M\}$ is the automaton composed by identifying in the obvious way the actions for processes $0, \ldots, n$ and the memory M. A process i is said to *participate* in an execution of a protocol if the execution contains a $start(v)_i$ action. The set of participating processes is called the execution's *participating set*. Note that this definition of the participating set matches our earlier definition of a participating set of input vectors. 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 i) has a unique *start* action (generated externally to the protocol) which precedes any *call* or *return* actions, alternates matching *call* and *return* actions, and has at most one *decide* action. We restrict our attention to well-formed executions.

2.4. Solvability. We are interested in solvability in the face of arbitrary fail-stop failures [17] (such failures also model processes being arbitrarily delayed or halted). To capture the notion of processes having fail-stop failures, we add to the process automaton a unique $fail(i)$ event. A process's execution is thus a sequence of actions ending in either a *decide* or a *fail* action. If the execution ended in a *fail* action the process is said to be *faulty*. An execution is *t-faulty* if up to t processes become faulty.

DEFINITION 2.11. *A protocol solves a decision task in an execution if the following condition holds. Let $\{i | i \in U\}$ be the processes that have start actions, and let $\{u_i | i \in U\}$ be their arguments. Let $\{j | j \in V\}$, $V \subseteq U$, be the processes that execute decide actions, 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 decide or fail action, and*
2. $\vec{O}$ *is a prefix of some vector in $\Delta(\vec{I})$.*

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*.[1]

2.5. One-shot IS. Our memory model is based on Borowsky and Gafni's *IS object* [11], a model that has proven to be a useful building block for the construction and analysis of protocols in many asynchronous, distributed systems [11, 12, 24, 25, 26, 31].

Informally, an $n + 1$-process IS object consists of a shared $n + 1$-dimensional memory array and supports a single type of operation, called *writeread*. Each *writeread* operation writes a value to a single shared memory array cell, and returns a "snapshot" view of the entire array in the state immediately following the write—hence the name "immediate snapshot." A *writeread* operation by process i writes its value to the ith cell of the memory array.
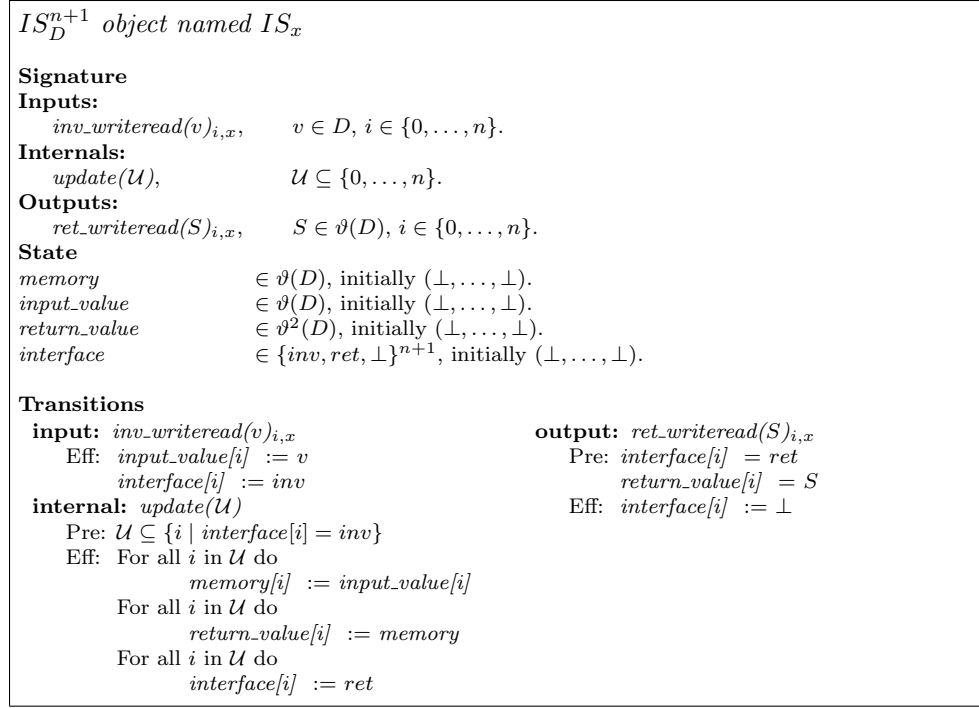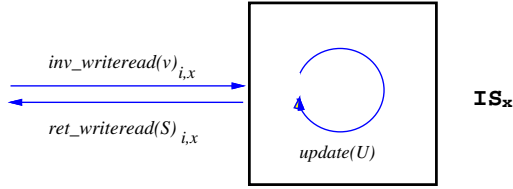
Formally, we can specify IS objects as I/O automata [28]. Let D be any data type, and define $\vartheta(D)$ to be the data type $(D \cup \{\perp\})^{n+1}$, the set of all $n + 1$-arrays each of whose cells contains either an element of D or $\perp$. We index the elements of $\vartheta(D)$ using the numbers in $\{0, \ldots, n\}$ and define $\vartheta^k(D)$ for $k \geq 2$ as $\vartheta(\vartheta^{k-1}(D))$, that is, a recursively growing vector of vectors. An IS automaton for $n + 1$-processes and data type D called IS_x is defined as in Figure 3. We refer to such an object as an IS_D^{n+1} object. When the data type and number of processes are clear from the context, we usually omit the subscripts and superscripts above.

In Figure 3, the operation $writeread(v, S)_{i,x}$ by process i on IS_x writes the value v to the ith cell of *memory* and subsequently returns a snapshot S. The idea of the automaton specification is to capture the notion of an update of a memory array location followed immediately by a snapshot view of the entire array. Using a style similar to that of the atomic snapshot memory specification of [1], we record the history of invocations and responses using interface variables and allow the combined "write and snapshot" operation itself to occur via an internal automaton transition at some point between the invocation and associated response. Figure 4 shows a stylized diagram of the IS object IS_x.

For all i, the $inv\_writeread(v)_{i,x}$ action simply writes the input value v to the ith cell of the *input_value* array of IS_x. This array provides temporary storage for inputs to the IS_D^{n+1} object. At the same time, the flag *"inv"* is written to the ith cell of the *interface* array, which indicates an input by process i. The *update(U)* action is the internal transition that periodically copies a set of values corresponding to the indexes in $\mathcal{U}$ from the *input_value* array to the *memory* array. The set $\mathcal{U}$ must be a subset of the indexes i with the property that *interface[i] = inv*; in other words, these are operations that have been invoked by participating processes and have not yet been updated in memory. Additionally, a copy of the *memory* array is written to the ith cell of the *return_value* array for each $i \in \mathcal{U}$. This corresponds to an IS view being collected. Finally, the flag *"ret"* is written to the ith cell of the *interface* array for each $i \in \mathcal{U}$, indicating that a response value to the invocation by process i is available.

---

[1] Note that we use the standard notion of *wait-free* protocols [27] and not the more restrictive notion of *bounded wait-free* protocols [27] even though, given that we consider deterministic algorithms and that in our discussion the input space is finite, one could actually place a bound on the length of any wait-free execution.

IS_D^{n+1} *object named* IS_x

Signature
Inputs:
    $inv\_writeread(v)_{i,x},$ $v \in D,\ i \in \{0, \ldots, n\}.$
Internals:
    $update(\mathcal{U}),$ $\mathcal{U} \subseteq \{0, \ldots, n\}.$
Outputs:
    $ret\_writeread(S)_{i,x},$ $S \in \vartheta(D),\ i \in \{0, \ldots, n\}.$
State
$memory$ $\in \vartheta(D),$ initially $(\bot, \ldots, \bot).$
$input\_value$ $\in \vartheta(D),$ initially $(\bot, \ldots, \bot).$
$return\_value$ $\in \vartheta^2(D),$ initially $(\bot, \ldots, \bot).$
$interface$ $\in \{inv, ret, \bot\}^{n+1},$ initially $(\bot, \ldots, \bot).$

Transitions
input: $inv\_writeread(v)_{i,x}$ **output:** $ret\_writeread(S)_{i,x}$
    Eff: $input\_value[i] := v$ Pre: $interface[i] = ret$
           $interface[i] := inv$ $return\_value[i] = S$
internal: $update(\mathcal{U})$ Eff: $interface[i] := \bot$
    Pre: $\mathcal{U} \subseteq \{i \mid interface[i] = inv\}$
    Eff: For all i in $\mathcal{U}$ do
               $memory[i] := input\_value[i]$
        For all i in $\mathcal{U}$ do
               $return\_value[i] := memory$
        For all i in $\mathcal{U}$ do
               $interface[i] := ret$

FIG. 3. *I/O automaton for an* IS_D^{n+1} *object with name* IS_x.

FIG. 4. *Diagram of* IS_x.

The $ret\_writeread(S)_{i,x}$ output action provides a response to a previous invocation by process i. Its only effect on the IS object is to reset the value $interface[i]$ to $\bot$, thereby preventing more than one response to an invocation. The $ret\_writeread(S)_{i,x}$ action can occur only after a return value has been written to the ith cell of the *return_value* array and the flag *"ret"* has been written to the corresponding cell in the *interface* array.

In the remainder of this section, we will state and prove a few basic properties about IS objects. These properties will be useful later, when we prove the correctness of a topological framework for analyzing the complexity of protocols in models of computation that include multiple IS objects.

In this paper we will consider only a restricted class of executions of IS_x, called *one-shot executions*, in which each object has at most one invocation and at most one response by any process.

LEMMA 2.12. *For any two distinct actions update($\mathcal{U}$) and update($\mathcal{U}'$) in a one-shot execution* α *of* IS_x, *the index sets* $\mathcal{U}$ *and* $\mathcal{U}'$ *are disjoint.*

Proof. Suppose without loss of generality that $\mathcal{U}$ occurs before $\mathcal{U}'$, and suppose $i \in \mathcal{U}$. Immediately after the action *update($\mathcal{U}$)*, *interface[i]* is equal to *"ret."* Since we are considering a one-shot execution, *interface[i]* will not return to the value *"inv"* for the remainder of the execution. Hence, the precondition of the *update($\mathcal{U}'$)* action guarantees that $i \notin \mathcal{U}'$. $\square$

We can now define what we mean by concurrent operations of IS_x.

DEFINITION 2.13. *Two operations writeread($v_i, S_i)_{i,x}$ and writeread($v_j, S_j)_{j,x}$ in a one-shot execution α of IS_x are concurrent if there exists an action update($\mathcal{U}$) in α such that $i, j \in \mathcal{U}$.*

The proof of the following lemma is immediate by construction.

LEMMA 2.14. *Consider any operation writeread($v_i, S_i)_{i,x}$ in a one-shot execution α of IS_x. Then $S_i[i] = v_i$.*

The value returned by a *ret_writeread(S)$_{i,x}$* action is a one-dimensional array of type $\vartheta(D)$. In the following lemma, we prove that IS_x exhibits the property that the set of snapshots returned in a one-shot execution can be totally ordered by the prefix relation defined in Definition 2.3.

LEMMA 2.15. *Consider any two writeread operations in a one-shot execution α, writeread($v_i, S_i)_{i,x}$ and writeread($v_j, S_j)_{j,x}$. Either S_i is a prefix of S_j or S_j is a prefix of S_i.*

Proof. Suppose the values v_i and v_j are written to *memory* by the actions *update($\mathcal{U}_i$)* and *update($\mathcal{U}_j$)*, respectively.

If these actions are the same, that is, if $\mathcal{U}_i = \mathcal{U}_j$, the two operations *writeread(v_i, $S_i)_{i,x}$* and *writeread($v_j, S_j)_{j,x}$* are concurrent. In this case, the value of *memory* that is copied to *return_value[i]* is identical to the value copied to *return_value[j]*, since both are copied by the same *update($\mathcal{U}_i$)* action. It follows that $S_i = S_j$. Now suppose $\mathcal{U}_i \neq \mathcal{U}_j$, and suppose *update($\mathcal{U}_i$)* occurs after *update($\mathcal{U}_j$)*. Since no *memory* cells are ever reset, it follows that the *memory* version that is written to *return_value[j]* during *update($\mathcal{U}_j$)* is a prefix of the version that is written to *return_value[i]* during *update($\mathcal{U}_i$)*. Hence S_j is a prefix of S_i. The case where *update($\mathcal{U}_j$)* occurs after *update($\mathcal{U}_i$)* is similar, and in this case we have that S_i is a prefix of S_j. The lemma follows. $\square$

The next lemma concerns what is referred to in [11] as the *immediacy* property of *IS* objects. If a value written to *memory* by an invocation by process j is contained in a snapshot of an operation by process i, then the snapshot returned to process j is a prefix of that returned to process i. This corresponds to the informal notion of a *writeread* operation by j happening *before* a writeread operation by i.

LEMMA 2.16. *Consider any two writeread operations in a one-shot execution α, writeread($v_i, S_i)_{i,x}$ and writeread($v_j, S_j)_{j,x}$. If $S_i[j] \neq \bot$, then S_j is a prefix of S_i.*

Proof. Suppose the values v_i and v_j are written to *memory* by the actions *update($\mathcal{U}_i$)* and *update($\mathcal{U}_j$)*, respectively, and suppose $S_i[j] \neq \bot$. This implies that either v_j was written to *memory* during *update($\mathcal{U}_i$)*, in which case $\mathcal{U}_i = \mathcal{U}_j$, or the action *update($\mathcal{U}_j$)* occurred before *update($\mathcal{U}_i$)*. In either case, we have that S_j must be a prefix of S_i. $\square$

2.6. The NIIS model. Our *nonuniform iterated immediate snapshot (NIIS) model* is a variant of the iterated immediate snapshot (IIS) model, first used implicitly by Herlihy and Shavit [25, 26], and later formulated as a computation model by Borowsky and Gafni [12].

The IIS model assumes a bounded sequence $IS_D^{n+1}, IS_{\vartheta(D)}^{n+1}, IS_{\vartheta^2(D)}^{n+1}, \ldots, IS_{\vartheta^{k-1}(D)}^{n+1}$ of IS objects, denoted by $IS_1, IS_2, IS_3, \ldots, IS_k$, where $k > 0$. On a high level, the IIS

$inv\_writeread(v)_{i,1}$

$ret\_writeread(S)_{i,1}$

$inv\_writeread(v)_{i,x}$

$ret\_writeread(S)_{i,x}$
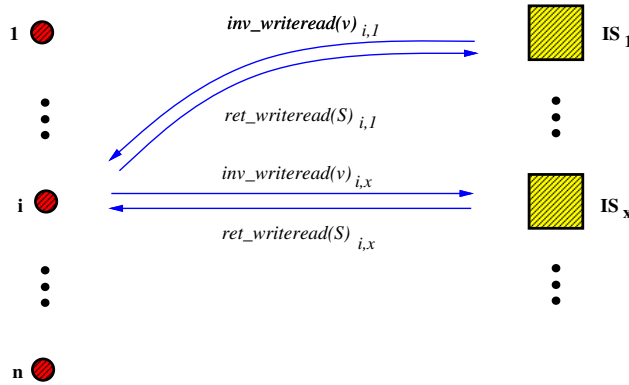
1

i

n

IS_1
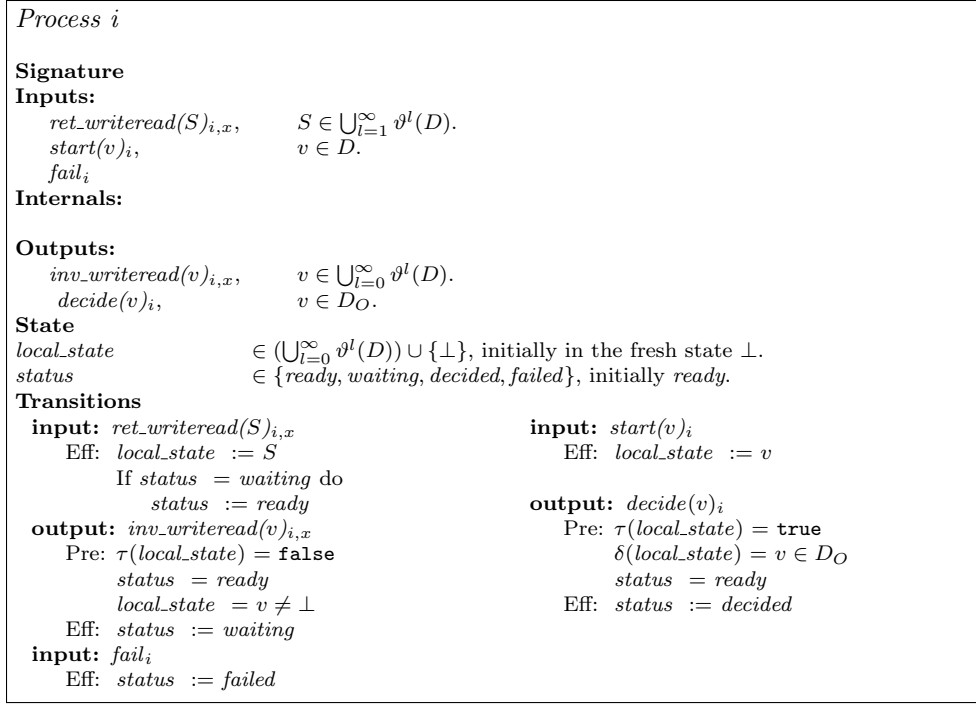
IS_x

FIG. 5. *Diagram of the NIIS model.*

model has each participating process proceed in accessing IS objects in ascending order in the sequence. On each object it executes a $writeread(v)_{i,1}$ action, with v equal to its *local_state*. The response from each object, consisting of an IS of its shared memory vector, is provided as input to the next object accessed in the sequence, until k objects have been accessed. Once a process has received an output from the kth IS object, it applies a decision map δ to this value to create its returned decision value. We note that there is no loss of generality in assuming a "full information model" where the input to one *writeread* operation is the response by the previous one.

We generalize the IIS model by introducing the *NIIS model*. Unlike IIS, the NIIS model assumes an unbounded sequence $IS_D^{n+1}, IS_{\vartheta(D)}^{n+1}, IS_{\vartheta^2(D)}^{n+1}, \ldots$ of IS objects, denoted $IS_1, IS_2, IS_3, \ldots$. A stylized interconnection diagram of the k-shot IIS model is given in Figure 5. The number of IS objects accessed by any two distinct processes in a given execution need not be the same, and, moreover, the number of objects accessed by any fixed process may vary from execution to execution. The motivation behind this is to be able to model complexity more accurately. In a given execution it may be the case that the necessary amount of computation will vary from process to process, from input value to input value, and indeed from execution to execution. This cannot be captured by a uniform model such as IIS in which every execution of a given protocol will involve the same number of steps.

The only significant difference between a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model and a protocol in the IIS model is that after each complete *writeread* operation, each process checks whether it has reached a final state by applying the predicate τ to the *local_state* variable. If τ returns `true`, the process executes a $decide(S)_i$ action and halts. Otherwise, it accesses the next IS object as in the IIS model, and so on. In fact, any protocol in the IIS model is equivalent to a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model, in which the predicate τ simply checks whether the *local_state* variable is of type $\vartheta^k(D)$.

Notice that, unlike the IIS model, the NIIS model permits unbounded length executions (assuming an unbounded number of IS objects) for some choices of the termination predicate map τ. However, we will consider only protocols for which τ is chosen such that the entire system does not have any infinite executions.

Each protocol in the NIIS model is fully characterized by the maximum number $n+1$ of processes that can participate, a predicate function $\tau : \bigcup_{l=0}^{\infty} \vartheta^l(D) \to \{\texttt{true}, \texttt{false}\}$, which each process applies to its *local_state* variable after each com-

---

Process i

Signature
Inputs:
    $ret\_writeread(S)_{i,x}$, $S \in \bigcup_{l=1}^{\infty} \vartheta^l(D)$.
    $start(v)_i$, $v \in D$.
    $fail_i$
Internals:

Outputs:
    $inv\_writeread(v)_{i,x}$, $v \in \bigcup_{l=0}^{\infty} \vartheta^l(D)$.
    $decide(v)_i$, $v \in D_O$.
State
$local\_state$ $\in (\bigcup_{l=0}^{\infty} \vartheta^l(D)) \cup \{\perp\}$, initially in the fresh state $\perp$.
$status$ $\in \{ready, waiting, decided, failed\}$, initially *ready*.
Transitions
  input: $ret\_writeread(S)_{i,x}$ **input:** $start(v)_i$
      Eff: $local\_state := S$ Eff: $local\_state := v$
         If $status = waiting$ do
            $status := ready$ **output:** $decide(v)_i$
  output: $inv\_writeread(v)_{i,x}$ Pre: $\tau(local\_state) = \mathtt{true}$
      Pre: $\tau(local\_state) = \mathtt{false}$ $\delta(local\_state) = v \in D_O$
          $status = ready$ $status = ready$
          $local\_state = v \neq \perp$ Eff: $status := decided$
      Eff: $status := waiting$
  input: $fail_i$
      Eff: $status := failed$

FIG. 6. *I/O automaton for process i running NIIS protocol.*

plete *writeread* operation to determine whether or not to decide, and a decision map $\delta : \bigcup_{l=0}^{\infty} \vartheta^l(D) \to D_O$, where D_O is an arbitrary data type, which we call the protocol's *output data type*. We refer to the protocol obtained by fixing these parameters as $\mathcal{P}_{(n,\tau,\delta)}$.

We specify each IS object as in Figure 3, and each process i as in Figure 6. The protocol can then be specified by composing the automata for the processes and the automata for IS objects by matching up invocations and responses from consecutive IS objects in the natural way. The resulting protocol automaton is denoted by $\mathcal{P}_{(n,\tau,\delta)} = \{0, 1, \ldots, n; IS_1, IS_2, \ldots\}$.

For any execution α of $\mathcal{P}_{(n,\tau,\delta)}$, the processes' input values can conveniently be represented using an $n + 1$-dimensional *input vector* $\vec{I}$, as specified in the previous section, with input data type D. The ith entry of $\vec{I}$ is the input of process i. Similarly, the processes' output values in α can be represented using an $n+1$-dimensional *output vector* $\vec{O}$. The ith entry of $\vec{O}$ is the output of process i.

It should be noted that the notions of participating processes and sets defined for executions and input vectors are consistent; a process i participates in an execution α if and only if the index i participates in the input vector $\vec{I}$ corresponding to α. Therefore, when the meaning is clear from the context, we usually omit qualifying a participating set with an execution or input vector.

We note that we have added $fail_i$ actions to the protocol to achieve the stopping failure property. We note that by construction if α is an execution of $\mathcal{P}_{(n,\tau,\delta)}$ that contains a $fail_i$ action, then α contains no actions locally controlled by i ($inv\_writeread(v)_{i,x}$ or $decide(S)_i$) after the $fail_i$ action.

2.7. Complexity measures for the IIS and NIIS models. We now define the complexity measures to be used for analyzing the performance of protocols in the NIIS model. Since the IIS model is equivalent to a special case of the NIIS model, these measures also apply directly to the IIS model.

Let $\mathcal{P}_{(n,\tau,\delta)}$ be a protocol in the NIIS model solving a given decision task $\mathcal{D}$, let $\vec{I}$ be an input vector, and let α be any execution of $\mathcal{P}_{(n,\tau,\delta)}$ that corresponds to $\vec{I}$. For all i, let t_i be the number of IS objects accessed by process i in α. We first define the time complexity of the execution α.

DEFINITION 2.17. *The time complexity of α, denoted t_α, is $\max_i t_i$, the maximum number of IS objects accessed by any process.*

We note that t_α is well defined, since the number of processes $n + 1$ is finite. Moreover, by definition of the max function, t_α is an integer value. We use the definition given above to define the time complexity of the protocol $\mathcal{P}$ on the input vector $\vec{I}$.

DEFINITION 2.18. *The time complexity of $\mathcal{P}_{(n,\tau,\delta)}$ on $\vec{I}$, denoted $t_{\vec{I}}$, is the supremum of the set $\{t_\alpha \mid t_\alpha$ is an execution corresponding to $\vec{I}\}$.*

Finally, we define the complexity of a protocol $\mathcal{P}_{(n,\tau,\delta)}$ on an input set I.

DEFINITION 2.19. *The time complexity of $\mathcal{P}_{(n,\tau,\delta)}$ on I, denoted t_I, is the supremum of the set $\{t_{\vec{I}} \mid \vec{I} \in I\}$.*

The reason for preferring these simple, discrete complexity measures over other, more elaborate, measures such as real time, for instance, is the highly regular structure of the IIS and NIIS models. We make the assumption that each access to an IS object takes the same amount of time and do not worry about breaking up the time required to complete each access to an IS object into subparts. Instead, we group the time spent on invocation, response, and on local computation at the IS object. This assumption is somewhat strong, as the presence of asynchrony in our model will tend to introduce varying delays for each access to an object. However, we believe that, as a first step toward a complexity theory, this assumption is justifiable, as it allows for complexity measures that are simple and easy to apply, and that have a particularly nice topological representation, as we will see in section 4.

3. A topological framework. In this section we first introduce some known tools from the field of algebraic topology and show how they may be used to model decision tasks and protocols in the NIIS model of computation. We then introduce a new tool for analyzing complexity in this setting, called the nonuniform iterated chromatic subdivision.

3.1. Basic topological definitions and concepts. This section introduces the basic topological definitions and concepts that we shall need for modeling decision tasks and wait-free protocols in the NIIS model. Some of these definitions are fairly standard and are mainly taken from popular textbooks on algebraic topology [30, 34], while others are due to Herlihy and Shavit [24, 25, 26]. The statements and proofs related to subdivisions are novel to this work. Some of the figures used in this section are also adopted from Herlihy and Shavit's work [24, 25, 26].

A *vertex* $\vec{v}$ is a point in a Euclidean space $\mathbb{R}^l$. A set $\{\vec{v}_0, \ldots, \vec{v}_n\}$ of vertexes is *geometrically independent* if and only if the set of vectors $\{\vec{v}_i - \vec{v}_0\}_{i=1}^n$ is linearly independent. Clearly, for a set of $n + 1$ vertexes to be geometrically independent, $l \geq n$. We can now define the concept of a geometric simplex, or simplex for short.

DEFINITION 3.1. *Let $\{\vec{v}_0, \ldots, \vec{v}_n\}$ be a geometrically independent set of vertexes in $\mathbb{R}^l$. We define the n-simplex S spanned by $\vec{v}_0, \ldots, \vec{v}_n$ to be the set of all points x*
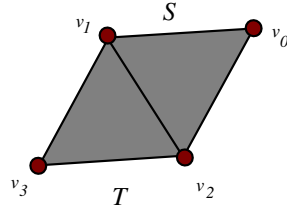
FIG. 7. *Example of a pure, two-dimensional simplicial complex.*

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 simplicity, we often denote the simplex spanned by a set $\{\vec{v}_0, \ldots, \vec{v}_n\}$ of geometrically 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 will sometimes include the number n as an explicit superscript when referring to a simplex; that is, we will write S^n to refer 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. The faces of S different from S itself are called the *proper* faces of S. The simplex spanned by the vertexes $\{\vec{v}_0, \vec{v}_1\}$ is a proper face of the 2-simplex S spanned by $\{\vec{v}_0, \vec{v}_1, \vec{v}_2\}$ in Figure 7.

The union of the proper faces of S is called the *boundary* of S and is denoted $Bd(S)$. The interior of S, denoted $\text{Int}(S)$, is defined by the set equation $\text{Int}(S) = S - Bd(S)$. For any set of points, the point

$$\vec{b} = \sum_{i=0}^{n} (\vec{v}_i/(n+1))$$

is their *barycenter*. The *barycenter* of a simplex S is the barycenter of its vertexes. In particular, if S is a vertex, then $\vec{b} = S$.

We will use a vertex to model the state of a single process, and a simplex to model consistent states of all the processes involved in solving a decision task or in running a protocol in the NIIS model. To model a collection of such states we need the concept of a geometric, simplicial complex, or complex for short, which is defined below.

DEFINITION 3.2. *A geometric simplicial complex* $\mathcal{K}$ *in the Euclidean space* $\mathbb{R}^l$ *is a collection of geometric simplexes in* $\mathbb{R}^l$ *such that*

- *every face* T *of every simplex* S *in* $\mathcal{K}$ *is contained in* $\mathcal{K}$, *and*
- *the intersection* U *of any two simplexes* S, T *in* K *is contained in* $\mathcal{K}$.

In this paper we will consider only finite complexes. The *dimension* of a complex $\mathcal{K}$, often denoted by $\dim(\mathcal{K})$, is the highest dimension of any of its simplexes and is also 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. All complexes considered in this paper are pure unless stated otherwise. A simplex S in $\mathcal{K}$ with dimension $\dim(S) = \dim(\mathcal{K})$ is called a *maximal* simplex.

Given a simplex S, let $\mathcal{S}$ denote the complex of all faces of S, and let $\dot{\mathcal{S}}$ denote the complex consisting of all proper faces of S. We note that, since $\dot{\mathcal{S}}$ contains all faces

of S except S itself, $\dim(\dot{S}) = \dim(S) - 1$. An example of a pure, two-dimensional simplicial complex, which we call $\mathcal{K}$, is shown in Figure 7. This complex equals the union of $\mathcal{S}$ and $\mathcal{T}$, where S is the 2-simplex spanned by $\{\vec{v}_0, \vec{v}_1, \vec{v}_2\}$, and T is the 2-simplex spanned by $\{\vec{v}_0, \vec{v}_1, \vec{v}_3\}$. Both S and T are maximal simplexes in this example.

If $\mathcal{L}$ is a subcollection of simplexes in $\mathcal{K}$ that is closed under containment and intersection, where $\dim(\mathcal{L}) \leq \dim(\mathcal{K})$, then $\mathcal{L}$ is a complex in its own right. It is called a *subcomplex* of $\mathcal{K}$. For example, the complex $\dot{S}$ of faces of S is a subcomplex of $\mathcal{K}$ in Figure 7.

One subcomplex of a complex $\mathcal{K}$ of particular interest is the subcomplex of all simplexes in $\mathcal{K}$ of dimension at most p, where p is some integer between 0 and $\dim(\mathcal{K})$. We call this subcomplex the *pth skeleton* of a $\mathcal{K}$, denoted $skel^p(\mathcal{K})$. The elements of the collection $skel^0(\mathcal{K})$ are called the *0-simplexes* of $\mathcal{K}$. The 0-skeleton of the complex $\mathcal{K}$ in Figure 7 is the collection of 0-simplexes $\{(\vec{v}_0), (\vec{v}_1), (\vec{v}_2), (\vec{v}_3)\}$. Similarly, the 1-skeleton of $\mathcal{K}$ is the union of the 0-skeleton described above and the collection $\{(\vec{v}_0, \vec{v}_1), (\vec{v}_0, \vec{v}_2), (\vec{v}_1, \vec{v}_2), (\vec{v}_1, \vec{v}_3), (\vec{v}_2, \vec{v}_3)\}$.

Let $|\mathcal{K}|$ be the subset $\bigcup_{S \in \mathcal{K}} S$ of $\mathbb{R}^l$ that is the union of the simplexes of $\mathcal{K}$. Giving each simplex its natural topology as a subspace of $\mathbb{R}^l$, 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 *polytope* of $\mathcal{K}$. Conversely, $\mathcal{K}$ is called a *triangulation* of $|\mathcal{K}|$.

In practice, the geometric representations we have given for simplexes and complexes are not always convenient, since the analytic geometry involved can get quite involved. Therefore, we introduce the notions of *abstract simplexes* and *abstract complexes*.

DEFINITION 3.3. *An abstract simplex S is a finite, nonempty set.*

The *dimension* of S is its cardinality. Each nonempty subset T of S is called a *face* of S. Each element of S is called a *vertex* of S. There is a close relationship between geometric simplexes and abstract simplexes. Any geometrically independent set of vectors $\{\vec{v}_0, \ldots, \vec{v}_n\}$ not only spans a geometric simplex; it also forms an abstract simplex.

DEFINITION 3.4. *An abstract complex $\mathcal{K}_a$ is a collection of abstract simplexes, such that if S is in $\mathcal{K}_a$, so is any face of S.*

Most concepts defined for geometric complexes immediately carry over to abstract complexes; the *dimension* of $\mathcal{K}_a$, often denoted by $\dim(\mathcal{K}_a)$, is the highest dimension of any of its simplexes. An n-dimensional abstract complex (or n-complex) is *pure* if every simplex is a face of some n-simplex. If $\mathcal{L}_a$ is a subcollection of $\mathcal{K}_a$ that is itself an abstract complex, then $\mathcal{L}_a$ is called a *subcomplex* of $\mathcal{K}_a$.

DEFINITION 3.5. *Let $\mathcal{K}$ be a geometric complex, and let V be the vertex set of $\mathcal{K}$. Let $\mathcal{K}_a$ be the abstract complex of all subsets S of V such that S spans a simplex in $\mathcal{K}$. Then $\mathcal{K}_a$ is called the* vertex scheme *of $\mathcal{K}$.*

DEFINITION 3.6. *Two abstract complexes $\mathcal{K}_a$ and $\mathcal{L}_a$ are* isomorphic *if there is a bijective correspondence ψ between their vertex sets such that a set S of vertexes is in $\mathcal{K}_a$ if and only if $\psi(S) \in \mathcal{L}_a$. The bijective correspondence ψ is called an* isomorphism.

THEOREM 3.7. *Every abstract complex $\mathcal{K}_a$ is isomorphic to the vertex scheme of some geometric complex $\mathcal{K}$ in $\mathbb{R}^{2\dim(\mathcal{K}_a)+1}$.*

We will not prove this theorem here. For a proof, see any standard textbook on algebraic topology [30, 34]. In the rest of this paper, for convenience, we will often use abstract and geometric representations of simplexes and complexes interchangeably.

We now define a way of "adding" simplexes, known as *starring*.
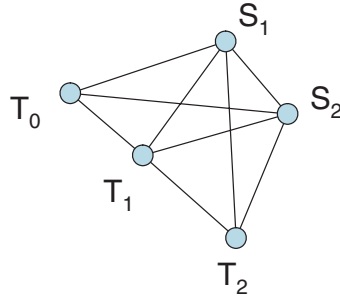
FIG. 8. *The star of two complexes S and T.*

DEFINITION 3.8. *Let $S = (s_0, \ldots, s_p)$ and $T = (t_0, \ldots, t_q)$ be simplexes whose combined sets of vertexes are affinely independent. Then the* star *of S and T, denoted $S \star T$, is the simplex $(s_0, \ldots, s_p, t_0, \ldots, t_q)$.*

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

DEFINITION 3.9. *Let $\mathcal{K}$ and $\mathcal{L}$ be simplicial complexes, not necessarily of the same dimension. Then the* star *of $\mathcal{K}$ and $\mathcal{L}$, denoted $\mathcal{K} \star \mathcal{L}$, is the collection of simplexes $\mathcal{K} \cup \mathcal{L} \cup \{S \star T \mid S \in \mathcal{K},\, T \in \mathcal{L}\}$.*

The star of two complexes $\mathcal{K}$ and $\mathcal{L}$ is a complex in its own right [30]. Figure 8 shows a complex consisting of two 3-simplexes and all their faces resulting from starring the complex S which includes the 1-simplex (s_0, s_1) and all its faces, and the complex T consisting of the two 1-simplexes (t_0, t_1) and (t_1, t_2) and all their faces.

The remainder of this section, however, which introduces a number of important topological concepts, such as simplicial maps, subdivisions, and carriers, is set in the context of geometric complexes.

We first define the notions of simplicial vertex maps and simplicial maps from one complex into another.

DEFINITION 3.10. *Let $\mathcal{K}$ and $\mathcal{L}$ be complexes, possibly of different dimensions, and let $\mu : skel^0(\mathcal{K}) \to skel^0(\mathcal{L})$ be a function mapping vertexes to vertexes. Suppose that whenever the vertexes $\vec{v}_0, \ldots, \vec{v}_n$ of $\mathcal{K}$ span a simplex of $\mathcal{K}$, the vertexes $\mu(\vec{v}_0), \ldots, \mu(\vec{v}_n)$ span a simplex of $\mathcal{L}$. Then μ is called a* simplicial vertex map *from $\mathcal{K}$ to $\mathcal{L}$. μ can be extended to a continuous map $\mu_* : |\mathcal{K}| \to |\mathcal{L}|$ such that*

$$x = \sum_{i=0}^{n} t_i \vec{v}_i \Rightarrow \mu_*(x) = \sum_{i=0}^{n} t_i \mu_*(\vec{v}_i).$$

This continuous extension is called a simplicial map from $\mathcal{K}$ to $\mathcal{L}$.

For simplicity, we henceforth refer to the simplicial vertex map μ as the simplicial map, without actual reference to the continuous extension μ_*, which is less relevant for our purposes. As a further abuse of notation, we usually write $\mu : \mathcal{K} \to \mathcal{L}$ when we refer to the simplicial vertex map, glossing over the fact that this map is in fact defined only on the vertexes of $\mathcal{K}$, and that the image of the map is a subset of the vertex set of $\mathcal{L}$. Henceforth, unless stated otherwise, all maps between complexes are assumed to be simplicial. An example of a simplicial map is given in Figure 9.

We note that a simplex and its image under a simplicial map need not have the same dimension. A simplicial map $\mu : \mathcal{K} \to \mathcal{L}$ is *noncollapsing* if it preserves dimension; that is, for all $S \in \mathcal{K}$: $\dim(\mu(S)) = \dim(S)$.
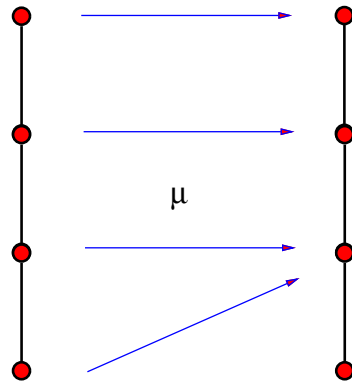
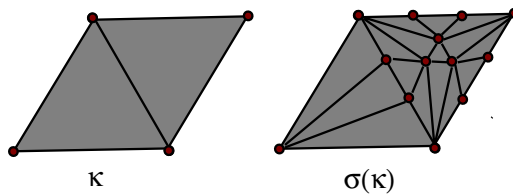FIG. 9. *Example of a simplicial map between two complexes.*

FIG. 10. *Example of a pure, two-dimensional simplicial complex and a subdivision of it.*

DEFINITION 3.11. *A* coloring *of an n-dimensional complex $\mathcal{K}$ is a noncollapsing simplicial map $\chi : \mathcal{K} \to \mathcal{S}$, where $\mathcal{S}$ is an n-simplex.*

Intuitively, a coloring corresponds to a labeling of the vertexes of the complex such that no two neighboring vertexes (connected by a 1-simplex) have the same label. A *chromatic complex* $(\mathcal{K}, \chi)$ is a complex $\mathcal{K}$ together with a coloring χ of $\mathcal{K}$. When it is clear from the context, we specify the chromatic complex $(\mathcal{K}, \chi)$ simply as the complex $\mathcal{K}$, omitting explicit mention of the coloring χ.

DEFINITION 3.12. *Let $(\mathcal{K}, \chi_{\mathcal{K}})$ and $(\mathcal{L}, \chi_{\mathcal{L}})$ be chromatic complexes, and let $\mu : \mathcal{K} \to \mathcal{L}$ be a simplicial map. We say that μ is* chromatic *if, for every vertex $\vec{v} \in \mathcal{K}$, $\chi_{\mathcal{K}}(\vec{v}) = \chi_{\mathcal{L}}(\mu(\vec{v}))$.*

In other words, μ is chromatic if it maps each vertex in $\mathcal{K}$ to a vertex in $\mathcal{L}$ of the same color. All the simplicial maps we consider in this paper are chromatic. We can now define the concepts of a *subdivision* of a complex and the *carrier* of a simplex in a subdivision.

DEFINITION 3.13. *Let $\mathcal{K}$ be a complex in $\mathbb{R}^l$. A complex $\sigma(\mathcal{K})$ is said to be a* subdivision *of $\mathcal{K}$ if the following two conditions hold:*

- *Each simplex in $\sigma(\mathcal{K})$ is contained in a simplex in $\mathcal{K}$.*
- *Each simplex of $\mathcal{K}$ equals the union of finitely many simplexes in $\sigma(\mathcal{K})$.*

An example of a complex and its subdivision is given in Figure 10.

DEFINITION 3.14. *If S is a simplex of $\sigma(\mathcal{K})$, the* carrier *of S, denoted $carrier(S)$, is the unique smallest $T \in \mathcal{K}$ such that $S \subset T$.*

The concept of a carrier of a simplex is illustrated in Figure 11. The original

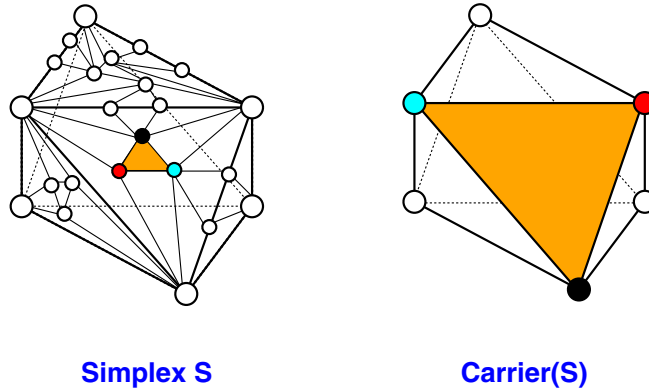**Simplex S**   **Carrier(S)**

FIG. 11. *The carrier of a simplex.*

complex is shown on the right, and the subdivided complex is shown on the left. A simplex S in the subdivision and the corresponding carrier $carrier(S)$ in the original complex are highlighted in the figure.

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 $\sigma(\mathcal{K})$, $\chi_{\sigma(\mathcal{K})}(S) \subseteq \chi_\mathcal{K}(carrier(S))$. 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 $S \in \sigma_1(\mathcal{K})$, $carrier(S) = carrier(\mu(S))$. All subdivisions we consider in this paper will be chromatic, unless explicitly stated otherwise.

3.2. Topological modeling of decision 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. To illustrate our constructions, we will first explain on a high level how to topologically model tasks, specifically the well-known Unique-Id task of Example 2.9. We will then provide detailed topological definitions of decision tasks.

We represent all possible input vectors to a task as a simplicial complex. In the case of the *Unique-Id* task, there is a (unique) $n + 1$-dimensional input vector $\vec{I} = [0, \dots, 0]$ represented as a simplex S, with dimension $0 \leq \dim(S) \leq n$. The dimension of S equals the number of non-$\perp$ elements in the vector.

From here on, each vertex $\vec{v}$ in a simplex S will be labeled with a process id and an input value. We will use $ids(S)$ to denote a simplex S's set of process ids (similarly for a complex), and $vals(S)$ to denote the multiset of values in S (similarly for a complex). If $\vec{J}$ is a prefix of $\vec{I}$, then the simplex corresponding to $\vec{J}$ is a face of S. The set I of input vectors is thus modeled as a complex $\mathcal{I}$ of input simplexes, called the *input complex*. For the *Unique-Id* task each vertex in $\vec{I}$ is labeled $\langle i, v_i \rangle$, where $\vec{I}[i] = v_i = 0$.

Similarly, the set O of output vectors is modeled as a complex $\mathcal{O}$ of output simplexes, called the *output complex*. In the *Unique-Id* task we represent each $n + 1$-dimensional output vector $\vec{O} = [x_1, \dots, x_n]$, where for all i, j, either $x_i = \perp$ or $0 \leq x_i \leq n$ and $(x_i = x_j) \Rightarrow (x_i = \perp)$, as a simplex T, with dimension $0 \leq \dim(T) \leq n$. Each vertex $\vec{v}$ in T is labeled with a process id and an output value $\langle i, v_i \rangle$, where $\vec{O}[i] = v_i$. As before, if $\vec{P}$ is a prefix of $\vec{O}$, then the simplex corresponding to $\vec{P}$ is a face of T.

A *topological task specification map* Γ maps the input complex to the output complex in a way that captures the input/output vector relation γ of a given task. The *Unique-Id* task induces a topological task specification map Γ in the natural way, mapping each input simplex $S \in \mathcal{I}$ to a set $\Gamma(S)$ of output simplexes in $\mathcal{O}$, with the property that for all $T \in \Gamma(S)$, the set *vals*(T) contains no non-$\perp$ duplicates.

We can now give an alternative, topological representation of the *Unique-Id* decision task by simply specifying it as a tuple $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ consisting of an input complex $\mathcal{I}$, an output complex $\mathcal{O}$, and a topological task specification map Γ.

This topological representation gives an alternative interpretation of the notion of "similar" system states. The processes corresponding to vertexes on the common boundary of the two simplexes cannot distinguish between the two global output sets based on their own output values. Unlike graph-theoretic models (e.g., [7]), simplicial complexes capture in a natural way the notion of the *degree* of similarity between the two global output sets: it is the dimension of the intersection of the two 2-simplexes.

We now give a formal procedure for how to specify a given decision task $\mathcal{D} = \langle I, O, \gamma \rangle$ topologically. We first construct a representation using abstract simplexes and complexes. It then follows from Theorem 3.7 that there exists a representation using geometric simplexes and complexes for which the vertex scheme is isomorphic to the abstract representation. There are standard ways of constructing such geometric complexes [30], but we choose not to get into the details of these constructions in this paper.

DEFINITION 3.15. *Let* $\vec{I} \in I$ *be an input vector. The* input simplex *corresponding to* $\vec{I}$, *denoted* $S(\vec{I})$, *is the abstract simplex* $(\langle i_0, v_{i_0} \rangle, \ldots, \langle i_m, v_{i_m} \rangle)$, *where* $v_{i_j} = \vec{I}[i_j]$, *and where for all* i_j, $i_0 \leq i_j \leq i_m$, $i_j \in \{0, \ldots, n\} \wedge \vec{I}[i_j] \neq \perp$, *and for all* i, $(\vec{I}[i] = \perp) \Rightarrow (i \notin \{i_0, \ldots, i_m\})$.

DEFINITION 3.16. *Let* $\vec{O} \in O$ *be an output vector. The* output simplex *corresponding to* $\vec{O}$, *denoted* $T(\vec{O})$, *is the abstract simplex* $(\langle i_0, v_{i_1} \rangle, \ldots, \langle i_m, v_{i_m} \rangle)$, *where* $v_{i_j} = \vec{O}[i_j]$, *and where for all* i_j, $i_0 \leq i_j \leq i_m$, $i_j \in \{0, \ldots, n\} \wedge \vec{O}[i_j] \neq \perp$, *and for all* i, $(\vec{O}[i] = \perp) \Rightarrow (i \notin \{i_0, \ldots, i_m\})$.

In other words, the vertexes in an input/output simplex correspond exactly to the non-$\perp$ values of the input/output vectors. Having defined input and output simplexes, we can define input and output complexes.

DEFINITION 3.17. *The* input complex *corresponding to* I, *denoted* $\mathcal{I}$, *is the collection of input simplexes* $S(\vec{I})$ *corresponding to the input vectors of* I.

DEFINITION 3.18. *The* output complex *corresponding to* O, *denoted* $\mathcal{O}$, *is the collection of output simplexes* $T(\vec{O})$ *corresponding to the output vectors of* O.

Definitions 3.17 and 3.18 make sense topologically due to the following lemma which follows from the fact that the sets of input and output vectors we consider are prefix-closed (see Definition 2.5).

LEMMA 3.19. *Given a set* I *of input vectors (alternatively a set* O *of output vectors), the corresponding input complex* $\mathcal{I}$ *(output complex* $\mathcal{O}$*), as defined in Definition* 3.17, *is an abstract, chromatic complex.*

Given a pair of (abstract) input and output complexes, we may apply Theorem 3.7 to construct a corresponding pair of geometric chromatic input and output complexes by embedding the abstract complexes in $\mathbb{R}^{2n+1}$. As discussed in section 3.1, we will

thus work with both interchangeably in the remainder of this paper.

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

DEFINITION 3.20. *The* topological task specification map *corresponding to γ, denoted $\Gamma \subseteq \mathcal{I} \times \mathcal{O}$, is defined as follows:*

$$(S(\vec{I}), T(\vec{O})) \in \Gamma \iff (\vec{I}, \vec{O}) \in \gamma.$$

As a convenient notation, for all $S(\vec{I}) \in \mathcal{I}$, we denote the set of simplexes $T(\vec{O})$ in $\mathcal{O}$ such that $(S(\vec{I}), T(\vec{O})) \in \Gamma$ by $\Gamma(S(\vec{I}))$. Usually, we simply refer to a topological task specification map as a "task specification map." We now prove that task specifications are id-preserving; if a process i has an input value, it must also have an output value, and vice versa.

LEMMA 3.21. *For all $S(\vec{I}) \in \mathcal{I}$, and all $T(\vec{O}) \in \Gamma(S(\vec{I}))$, $ids(T) = ids(S)$.*

Proof. Let $S(\vec{I})$ be any simplex in $\mathcal{I}$, and let $T(\vec{O}) \in \Gamma(S(\vec{I}))$. Then $\vec{O} \in \gamma(\vec{I})$ by Definition 3.20. Suppose $i \notin ids(S(\vec{I}))$. Then $\vec{I}[i] = \bot$ by Definition 3.15, and hence by Definition 2.6, $\vec{O}[i] = \bot$. It follows from Definition 3.16 that $i \notin ids(T(\vec{O}))$. Now suppose $i \notin ids(T(\vec{O}))$. Then $\vec{O}[i] = \bot$ by Definition 3.16, and hence by Definition 2.6, $\vec{I}[i] = \bot$. It follows from Definition 3.15 that $i \notin ids(S(\vec{I}))$. $\square$

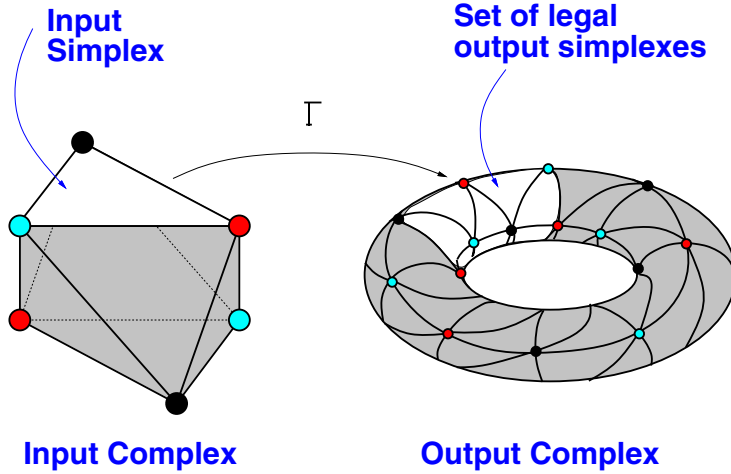A schematic illustration of a topological decision task specification is given in Figure 12.

FIG. 12. *A decision task.*

DEFINITION 3.22. *Given a decision task $\mathcal{D} = \langle I, O, \gamma \rangle$, the corresponding* topological representation *of the task, denoted $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$, consists of an input complex $\mathcal{I}$ corresponding to I, an output complex $\mathcal{O}$ corresponding to O, and a task specification map Γ corresponding to γ.*

In the remainder of this paper, we will specify decision tasks using Definitions 2.7 and 3.22 interchangeably. A set of inputs or outputs may thus be specified as either a vector or a simplex, the vertexes of which are labeled with process ids and values.

3.3. Topological modeling of NIIS protocols. We model protocols in the NIIS model in much the same way that we model decision tasks. As discussed in section 2, the sets of inputs and outputs for any execution α of a protocol $\mathcal{P}_{(n,\tau,\delta)}$

in the NIIS model can be modeled using $n + 1$ process input and output vectors. We denote the sets of input vectors and output vectors of a protocol by I and O, respectively. We are interested only in protocols that solve decision tasks, so we may assume that the set I of possible input vectors to a protocol is prefix-closed. The following lemma states that for any protocol in the NIIS model, the set O of possible output vectors from all executions of the protocol must necessarily be prefix-closed. Recall from section 2.6 that we are considering only the set of fair (and hence finite) executions of a protocol here.

LEMMA 3.23. *Let O be the set of possible output vectors of a protocol $\mathcal{P}_{(n,\tau,\delta)}$ in the NIIS model, with a corresponding set of input vectors I. Then O is prefix-closed.*

Proof. Let $\vec{O}$ be an output vector produced by the execution α_O, and let $\vec{P}$ be a prefix of $\vec{O}$. We construct an execution α_P as follows: For each i such that $\vec{O}[i] = v_i \neq \perp = \vec{P}[i]$, replace the action *decide(S)$_i$* (S is the output value returned by that action) in α_O with a *fail$_i$* action, meaning that process i fail-stopped before deciding. Clearly, the execution thus obtained is a possible execution of $\mathcal{P}_{(n,\tau,\delta)}$, and its output vector is $\vec{P}$. Hence $\vec{P}$ is in O, and O is prefix-closed. □

Given that both the set of input vectors I and the set of output vectors O associated with a protocol $\mathcal{P}_{(n,\tau,\delta)}$ are prefix-closed sets of vectors, we can construct corresponding input and output complexes, denoted $\mathcal{I}$ and $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$, respectively. These complexes are constructed in the same way as the complexes corresponding to input and output sets of vectors for decision tasks, and the proofs that they are indeed chromatic complexes are also identical. The output complex $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$ is called a *protocol complex.*

Let $\mathcal{J}$ be a subcomplex of the input complex $\mathcal{I}$. The set of possible outputs when the protocol is given inputs corresponding to simplexes in $\mathcal{J}$ is denoted $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$.

LEMMA 3.24. *Let $\mathcal{J}$ be a subcomplex of $\mathcal{I}$. Then $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$ is a subcomplex of $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$.*

Proof. It suffices to show that $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$ is a complex, since $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$ is clearly a subset of $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$. Consider the set of vectors J corresponding to the subcomplex $\mathcal{J}$ as the set of input vectors to the protocol $\mathcal{P}_{(n,\tau,\delta)}$. This set is prefix-closed since $\mathcal{J}$ is a complex and hence is closed under containment. Hence the set P of output vectors given input vectors in J is prefix-closed by Lemma 3.23. It follows that the complex corresponding to $\mathcal{P}_{(n,\tau,\delta)}$ with input complex $\mathcal{J}$, denoted $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{J})$, is by construction a complex and hence a subcomplex of $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{I})$. □

In the remainder of this paper, we will specify protocols in NIIS using both its formal specification from section 2.6 as well as protocol complexes as described in this section interchangeably. A set of inputs or outputs may thus be specified as either a vector or a simplex, the vertexes of which are labeled with process ids and values.

3.4. Subdivisions. The standard chromatic subdivision was introduced by Herlihy and Shavit as part of their work on asynchronous computability [24, 25, 26]. It is essentially a chromatic generalization of the standard barycentric subdivision from classical algebraic topology [30, 34]. In this section, we will present a complete, formal definition of the standard chromatic subdivision, together with a proof that it is, in the topological sense, a chromatic subdivision of a given complex. As noted earlier, such a proof also provides the necessary formal basis for the use of the standard chromatic subdivision in [12, 26]. We note that our definition is somewhat different from that of Herlihy and Shavit [24, 25, 26], as it is based on an explicit, inductive, geometric construction. We also introduce the concept of a *nonuniform chromatic subdivision*, a generalization of the standard chromatic subdivision, in which the different simplexes

of a complex are not necessarily subdivided the same number of times. Informally, a nonuniform chromatic subdivision of level 1 of a complex $\mathcal{K}$, denoted by $\widetilde{\mathcal{X}}^1(\mathcal{K})$, is constructed by choosing, for each n-simplex in $\mathcal{K}$, a *single* face of the simplex (a face can be of any dimension and can also be the whole simplex) to which we apply the standard chromatic subdivision. We then induce the subdivision onto the rest of the simplex. The subdivisions of any two intersecting simplexes must be such that they agree on their shared face. Examples can be seen in Figure 13. Its right-hand side shows a valid nonuniform chromatic subdivision of a complex where, for example, the simplex (b, c, d)'s subdivision is the result of subdividing the 1-face (c, d) once and then inducing this subdivision onto the rest of the simplex. The left-hand side structure is not a legal subdivision, since the subdivision of the simplex (b, c, d) does not agree with that of the simplex (a, b, d) on the shared face (b, d). This structure is not even a simplicial complex, since it contains an object that is not a simplex (the cross-hatched region in Figure 13). A kth level nonuniform chromatic subdivision of a complex $\mathcal{K}$, denoted by $\widetilde{\mathcal{X}}^k(\mathcal{K})$, is generated by repeating this process k times, where only simplexes in faces that were subdivided in round $k-1$ can be subdivided in phase k. The complex on the right-hand side of Figure 13 is an example of a nonuniform chromatic subdivision of level 2, since the face (a, d) is subdivided twice.
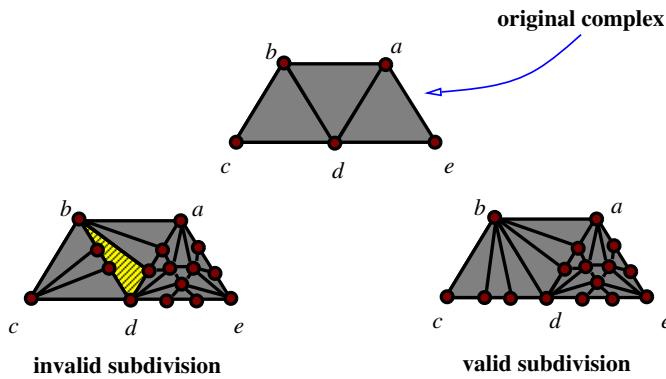

Fig. 13. *Valid and invalid nonuniform subdivisions.*

Later in this section we will show that the nonuniform chromatic subdivisions correspond in a natural way to the set of protocol complexes in the NIIS model of computation. As an execution in the NIIS model unfolds, some processes continue to step through IIS objects while others fail or decide. For a given input simplex, each transition through an IIS object by a subset of processes will correspond to a subdivision of the face corresponding to their respective vertexes. The other faces of the simplex, ones corresponding to the remaining processes, are not subdivided further, corresponding to the idea that the respective processes have either failed or decided. In the input complex, input simplexes sharing the same face have compatible subdivisions. As it turns out, each nonuniform standard chromatic subdivision is equal to some NIIS protocol complex (up to isomorphism).

3.4.1. The standard chromatic subdivision. In this section we provide our definition of the *standard chromatic subdivision* and prove that this definition does indeed specify a chromatic subdivision of a given complex.

Let $\mathcal{K}$ be a pure, n-dimensional, chromatic geometric complex, where the colors are the numbers in $0, \ldots, n$. Label each vertex $\vec{v}$ in $\mathcal{K}$ with $\langle i, v_i \rangle$, where i is the color

(process id) of $\vec{v}$ denoted also as $id(\vec{v})$, and v_i is a value (denoted also as $val(\vec{v})$) in some set D_I chosen such that no two vertexes in $\mathcal{K}$ have the same label. Note that two adjacent vertexes may have the same values if they have different colors. We define the standard chromatic subdivision of $\mathcal{K}$ by inductively defining a sequence of subdivisions $\mathcal{L}_p$ of the skeletons of $\mathcal{K}$, $0 \le p \le n$, as follows. We begin with the following auxiliary definition of the *containment conditions* among labeled vertexes.

DEFINITION 3.25. *For any set of simplexes $T = (\vec{t_0}, \ldots, \vec{t_r})$ in a complex $\mathcal{K}$ with labels $\langle i, S_i \rangle$, where S_i is the vertex scheme of some subset of simplexes in $\mathcal{K}$, define the* containment conditions *on the labels of T for any $1 \le i, j \le r$, $i \ne j$, as follows:*

C1. $id(\vec{t_i}) \ne id(\vec{t_j})$.

C2. $id(\vec{t_i}) \in ids(val(\vec{t_i}))$.

C3. $val(\vec{t_i})$ *is a face of* $val(\vec{t_j})$ *or vice versa.*

C4. $id(\vec{t_j}) \in ids(val(\vec{t_i})) \Rightarrow val(\vec{t_j})$ *is a face of* $val(\vec{t_i})$.

Condition C1 simply states that the given simplex is chromatic; that is, vertexes are colored with different ids. The remaining three conditions, which we will elaborate on shortly, will be used to capture the relation among the values written and read by a collection of *writeread* operations. In a nutshell, if one thinks of the $id(\vec{t_i})$ as the value written and the $ids(val(\vec{t_i}))$ as the IS value returned, then conditions C2, C3, and C4 correspond to the properties in Lemmas 2.14, 2.15, and 2.16.

We inductively define $\mathcal{L}_p$. Let $\mathcal{L}_0 = skel^0(\mathcal{K})$. Inductively assume that $\mathcal{L}_{p-1}$ is a chromatic subdivision of the $p-1$-skeleton of $\mathcal{K}$ where each vertex $\vec{v}$ in $\mathcal{L}_{p-1}$ is labeled $\langle i, S_i \rangle$, and where S_i is the vertex scheme of some simplex in $skel^{p-1}(\mathcal{K})$. We further assume that the labels $\langle i, S_i \rangle$ are such that any $T = (\vec{t_0}, \ldots, \vec{t_r})$, where $r \le p-1$, is a simplex in $\mathcal{L}_{p-1}$ if and only if $ids(T) \subseteq ids(carrier(T))$ and for all $1 \le i, j \le r$, $i \ne j$, the labels of T meet the containment conditions of Definition 3.25.

Figure 14 describes a simplex (S_0, S_1, S_2) whose $\mathcal{L}_0$ subdivision includes the black vertexes. It has been subdivided by $\mathcal{L}_1$, causing each simplex in $\mathcal{L}_0$ to be split in three by the two new vertexes in grey. Note that each of these pairs of vertexes has a different *id* but the same value field which represents the vertex scheme of its carrier $\mathcal{L}_0$ simplex. The reader can check that the labels of these vertexes meet the four containment conditions of Definition 3.25. If we think of the value fields of vertexes as representations of the return value of an IS *writeread* operation, then the four conditions capture the nature of two process executions in the IIS model. For any two processes, say, 0 and 1, there are three possible outcomes of passing through an IIS object, represented by the three simplexes of the subdivided $(\langle 0, 0 \rangle, \langle 1, 1 \rangle)$ simplex: 0 reads only itself and 1 reads both, 1 reads only itself and 0 reads both, or they both read each other.

Based on $\mathcal{L}_{p-1}$ we can now complete the definition of $\mathcal{L}_p$. Let $S = (\vec{s_0}, \ldots, \vec{s_p})$ be a p-simplex in $\mathcal{K}$. The set $Bd(S)$ is the polytope of a subcomplex of the $p-1$-skeleton of $\mathcal{K}$, and hence of a subcomplex of $\mathcal{L}_{p-1}$, which we denote $\mathcal{L}_{Bd(S)}$. Let $\vec{b}$ be the barycenter of S, and let δ be some positive real number such that $0 < \delta < 1/p$. For each $1 \le i \le p$, define $\vec{m_i}$ to be the point $(1 + \delta)\vec{b} - \delta\vec{s_i}$. These points are called the *midpoints* of S. Figure 14 shows the barycenter and midpoints of a 2-simplex. We can now label $\vec{m_i}$ with $\langle i, S \rangle$, S here being the vertex scheme of the geometric simplex S. Let M_S be the set of midpoints of S. We define $\mathcal{L}_S$ to be the union of $\mathcal{L}_{Bd(S)}$ and all the faces of all chromatic p-simplexes $T = (\vec{t_0}, \ldots, \vec{t_p})$, such that for all $1 \le i, j \le p : i \ne j$, $\vec{t_i} \in skel^0(\mathcal{L}_{Bd(S)}) \cup M_S$, and the four containment conditions of Definition 3.25 hold. $\mathcal{L}_p$ is thus the complex consisting of the union of the complexes $\mathcal{L}_S$, as S ranges over all the p-simplexes of $\mathcal{K}$.
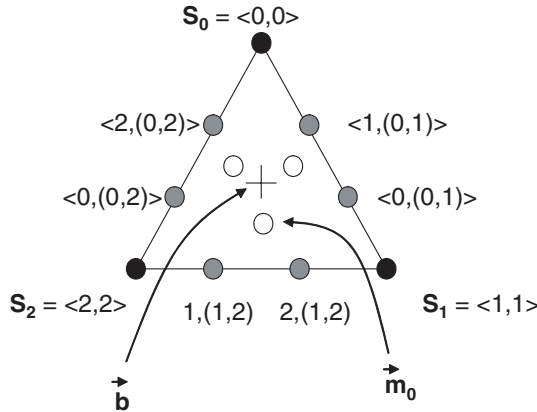
FIG. 14. *Example of the inductive step in the construction of the standard chromatic subdivision (for the sake of brevity, labels of simplexes slightly abuse notation).*
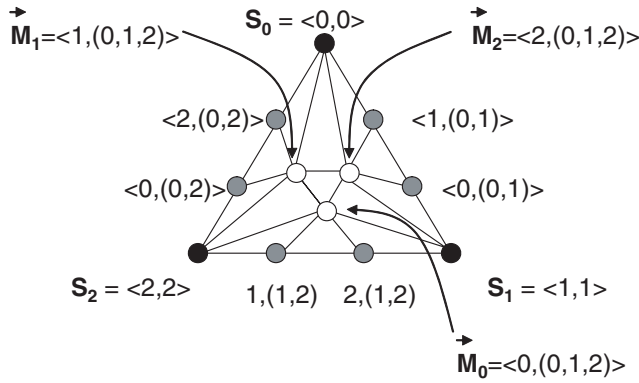

FIG. 15. *Example of the standard chromatic subdivision of a 2-simplex.*

Figure 15 describes a single subdivision $\mathcal{L}_p$ of a 2-simplex. Think of the value fields of vertexes as representations of the return values of an IS *writeread* operation by any one of three processes 1, 2, or 3. The complex captures all the possible outputs of executions in the IIS model. Each 2-simplex represents one such execution. A simplex with a face belonging to the boundary, such as the simplex $(\langle 1, 1 \rangle, \langle 2, (1, 2) \rangle, \langle 0, (0, 1, 2) \rangle)$ in the lower right-hand corner of Figure 15, represents an execution where process 1 executed a *writeread* reading of only itself, process 2 performed a *writeread* and read itself, and process 1, and finally process 0 performed a *writeread* and read all three. The middle simplex (m_0, m_1, m_2) corresponds to an execution where all three processes were concurrent and read each other's written value. The three containment conditions C2, C3, and C4 on the labels of the vertexes guarantee the properties of the IIS model as they are captured in Lemmas 2.14, 2.15, and 2.16: *writeread* operations are by different processes, *writeread*$_i$ returns i's written value, the returned snapshot by one process must include the values returned by the other, and finally, if i read j, then the value returned by j cannot contain an input that i did not read.

We now prove that this structure makes sense mathematically, that is, that it is in fact a subdivision of the p-skeleton of $\mathcal{K}$.

LEMMA 3.26. *For all $0 \leq p \leq n$, $\mathcal{L}_p$ is a chromatic subdivision of $skel^p(\mathcal{K})$.*

Proof. We argue by induction. The case $p = 0$ is trivial. So suppose $p > 0$, and suppose the claim holds for $\mathcal{L}_0, \ldots, \mathcal{L}_{p-1}$. We will first prove that $\mathcal{L}_p$ is a chromatic simplicial complex. To that end, we prove the following auxiliary lemma.

LEMMA 3.27. *For all p-simplexes S in $\mathcal{K}$, $\mathcal{L}_S$ is a chromatic complex.*

Proof. We must show that $\mathcal{L}_S$ is closed under containment and intersection. Let U be a simplex in $\mathcal{L}_S$, and let V be a face of U, where $0 \leq \dim(V) \leq \dim(U) < p$. If U is in $\mathcal{L}_{Bd(S)}$, then so is V, since $\mathcal{L}_{Bd(S)}$ is a complex (since $\mathcal{L}_p$ is a subdivision and hence a complex by assumption). Hence V is in $\mathcal{L}_S$. Suppose U is not contained in $\mathcal{L}_{Bd(S)}$. Then U must be the face of a p-simplex T as described above. By definition of $\mathcal{L}_S$, all the faces of T, and hence all faces of U, must be in $\mathcal{L}_S$. It follows that $\mathcal{L}_S$ is closed under containment.

Let U, V be simplexes in $\mathcal{L}_S$, and suppose their intersection, denoted by W, is nonempty. If U, V are both in $\mathcal{L}_{Bd(S)}$, it follows immediately that V^r is in $\mathcal{L}_{Bd(S)}$ and hence in L_S. Similarly, if U is in $\mathcal{L}_{Bd(S)}$ but V is not, then $W = U \cap V = U \cap (V \cap |\mathcal{L}_{Bd(S)}|)$. Note that $V \cap |\mathcal{L}_{Bd(S)}|$ is a simplex in $\mathcal{L}_{Bd(S)}$, since all the containment conditions of Definition 3.25 are satisfied. Hence it follows that W is in $\mathcal{L}_{Bd(S)}$, and hence in $\mathcal{L}_S$. If neither U nor V is in $\mathcal{L}_{Bd(S)}$, then since all faces of U and V are in $\mathcal{L}_S$, then so is W. It follows that $\mathcal{L}_S$ is closed under intersection, and hence is a complex. That $\mathcal{L}_S$ is chromatic follows from the fact that we include only chromatic simplexes in $\mathcal{L}_S$ in our construction (note that $\mathcal{L}_{p-1}$ and hence $\mathcal{L}_{Bd(S)}$ are chromatic by assumption).   □

Notice that for all distinct p-simplexes S, T we have that $|\mathcal{L}_S| \cap |\mathcal{L}_T| = S \cap T$, which is a simplex in $skel^{p-1}(\mathcal{K})$ and hence is the polytope of a subcomplex of $\mathcal{L}_{p-1}$ and hence of both $\mathcal{L}_S$ and $\mathcal{L}_T$. It follows that $\mathcal{L}_p$ is a simplicial complex [30]. It remains to show that $\mathcal{L}_p$ is a chromatic subdivision. To this end, we must first show that every simplex in $\mathcal{L}_p$ is contained in some simplex in $skel^p(\mathcal{K})$ and that every simplex in $skel^p(\mathcal{K})$ is the union of finitely many simplexes in $\mathcal{L}_p$. Now, it is clear from our construction that any simplex T_q in $\mathcal{L}_p$ is contained in some simplex S in $skel^p(\mathcal{K})$. Also, since for all simplexes S in $skel^p(\mathcal{K})$, the set of midpoints is finite, and $\mathcal{L}_{p-1}$ is a subdivision of $skel^{p-1}(\mathcal{K})$ by assumption, it follows that S is the union of finitely many simplexes in $\mathcal{L}_p$. Hence $\mathcal{L}_p$ is a subdivision. This subdivision is chromatic, since $\mathcal{L}_{p-1}$ is chromatic by assumption, since the colors used to color the midpoints of any simplex S are exactly the colors used to color S, and since any simplex in $\mathcal{L}_p$ including midpoints must satisfy the requirement that no two vertexes have the same color (id).   □

We are now ready to give our definition of the standard chromatic subdivision of a complex $\mathcal{K}$.

DEFINITION 3.28. *The* standard chromatic subdivision *of $\mathcal{K}$, denoted $\mathcal{X}(\mathcal{K})$, is the complex $\mathcal{L}_n$.*

An example of a complex and its standard chromatic subdivision is given in Figure 16. As one can see, the subdivision of a simplex as seen in Figure 15 is applied to all the simplexes in the complex $\mathcal{K}$. Applying the standard chromatic subdivision k times, where $k > 1$, yields a subdivision $\mathcal{X}^k(\mathcal{K}) = \mathcal{X}^{k-1}(\mathcal{X}(\mathcal{K}))$, which we call the kth iterated standard chromatic subdivision [24, 25, 26]. Since the standard chromatic subdivision of a complex is again a complex, and a chromatic subdivision of a chromatic subdivision of $\mathcal{K}$ is itself a chromatic subdivision of $\mathcal{K}$, $\mathcal{X}^k(\mathcal{K})$ is a chromatic subdivision of $\mathcal{K}$. The number k is called the *level* of the subdivision.

The following is the vertex scheme representation of the standard chromatic subdivision. This particularly compact formulation of the standard chromatic subdivision
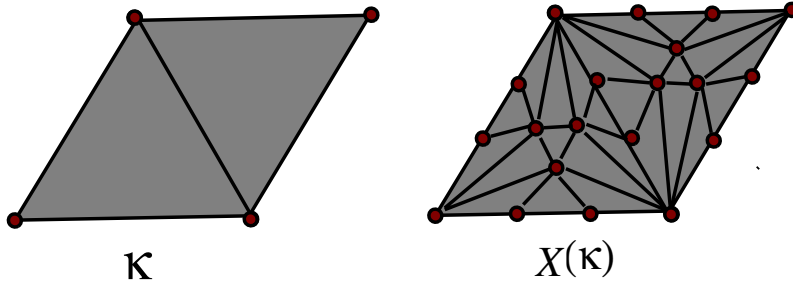
Fig. 16. *Example of a two-dimensional complex and its standard chromatic subdivision.*

is equivalent to the definition of Herlihy and Shavit [24, 25, 26].

LEMMA 3.29. *Let $\mathcal{K}$ be a pure, chromatic complex of dimension n. The vertex scheme of $\mathcal{X}(\mathcal{K})$ is the closure under containment of the set of all n-simplexes of the form $S = (\langle 0, S_0 \rangle, \ldots, \langle n, S_n \rangle)$, where for all i, S_i is the vertex scheme of some face of a simplex S in $\mathcal{K}$, and the following conditions hold for all $i \neq j$:*

- $i \in ids(S_i)$.
- S_i *is a face of S_j or vice versa.*
- *If $j \in ids(S_i)$, then S_j is a face of S_i.*

Furthermore, any abstract complex $\mathcal{L}$ with a vertex scheme meeting the above criteria has a realization as a geometric standard chromatic subdivision $\mathcal{X}(\mathcal{K}) = \mathcal{L}$ of the complex $\mathcal{K}$ induced by all vertexes $\langle i, S_i \rangle$ where $\dim(S_i) = 0$.

As before, the above three conditions have the exact same role as containment conditions C2, C3, and C4 of Definition 3.25.

Proof. We argue that the vertex scheme of $\mathcal{X}(\mathcal{K})$ meets the above properties by induction on k, where $0 \leq k \leq n$. It is immediate that the simplexes of $\mathcal{X}(\mathcal{K})$ lying in the subdivision $\mathcal{L}_0$ of $skel^0(\mathcal{K})$ are of this form (each such simplex is a vertex of $\mathcal{K}$ labeled with a process id and a value), and the three requirements of the lemma are all satisfied trivially.

Now suppose the claim holds for $0, \ldots, k-1$. Consider a simplex T lying in the subdivision $\mathcal{L}_k$ of $skel^k(\mathcal{K})$ and not in $\mathcal{L}_{k-1}$. Then $T = U \star V$, where U is a simplex in $\mathcal{L}_{k-1}$, and V is a simplex, each vertex of which is one of the midpoints in M_S, where $S = carrier(T)$. By assumption, V is nontrivial, meaning that there is at least one vertex in V. However, U may be trivial. For each vertex $\vec{v}$ in V, $val(\vec{v}) = S$, the vertex scheme of S. Hence, for i, j in $ids(V)$, since $ids(V) \subseteq ids(S)$, all the containment conditions of Definition 3.25 are met. For i, j in $ids(U)$, the conditions are satisfied by induction. Now suppose i is in $ids(U)$, while j is in $ids(V)$. Notice that $S_i = carrier(U)$, and $S_j = carrier(V) = S$.

The first condition follows by induction (for i) and since $ids(V) \subseteq ids(S) = vals(V)$ (for j). Since $carrier(U)$ is a face of S, it follows that S_i is a proper face of S, and hence of S_j, which equals the vertex scheme of S, and so the second condition is satisfied. It is clear that i is in $ids(S_j)$, since S_j equals the vertex scheme of S, and i is in $ids(carrier(U))$, which is a subset of $ids(S)$. That S_i is a face of S_j has already been established. It follows that, since $\mathcal{X}(\mathcal{K})$ is chromatic, $ids(U) \cap ids(V) = \emptyset$,

$ids(S_i) \subseteq ids(carrier(U))$, and j is not in $ids(carrier(U))$, j cannot be in $ids(S_i)$. It follows that the third condition is satisfied.

We now show that any abstract complex $\mathcal{L}$ with a vertex scheme meeting the above criteria has a realization as a geometric standard chromatic subdivision $\mathcal{X}(\mathcal{K}) = \mathcal{L}$ of the complex $\mathcal{K}$ induced by all vertexes with 0-dimensional labels, that is, $\langle i, S_i \rangle$ where $\dim(S_i) = 0$. We argue by induction on k, $0 \leq k \leq n$, the size of the set S_i in the label of any vertex $\langle i, S_i \rangle$ in $\mathcal{L}$. It is immediate that the vertexes with zero-dimensional labels meet the criteria since the first condition of Lemma 3.29 implies condition C2 and all other conditions of Definition 3.28 are satisfied trivially. These vertexes form $skel^0(\mathcal{K})$.

Now suppose the claim holds for $0, \ldots, k-1$. Consider the set of k-simplexes T_j, all having vertexes $\langle i, S_i \rangle$ out of the same subset of k ids in $\mathcal{L}$. By definition each simplex has at least one vertex whose label $\langle i, S_i \rangle$ has $\dim(S_i) = k$. By the induction hypothesis the realization of this set includes k complexes, each with a proper subset using $k-1$ of these ids, and each a geometric complex meeting the requirements of the geometric standard chromatic subdivision. These complexes which inductively form $skel^{k-1}$ meet each other at $k-2$-dimensional boundaries, and their union (topological sum [30]) is a complex that is a $k-1$-dimensional sphere (for example, three one-dimensional complexes, each a subdivision using two unique ids, form a one-dimensional sphere, i.e., a circle). Let $\vec{b}$ be the barycenter of this sphere, and let v_i, $0 \leq i \leq k$, be the set of vertexes in the sphere with zero-dimensional labels. Now, in the set of simplexes T_j there are by the definition of Lemma 3.29 k vertexes with k-dimensional labels. If we choose each of them as a midpoint $\vec{m}_i$ at some distance $(1 + \delta)\vec{b} - \delta\vec{v}_i$, from the barycenter $\vec{b}$ where $0 \leq \delta \leq 1/k$, then the simplexes T_j defined by the conditions of Lemma 3.29 form the interior of a simplex bounded by the $k-1$-dimensional sphere and fit the conditions of Lemma 3.26, implying that the above realization of $\mathcal{L}$ is a geometric standard chromatic subdivision of the complex $\mathcal{K}$ induced by the vertexes with zero-dimensional labels in $\mathcal{L}$.   □

In the remainder of this paper, we will usually work with this description of the standard chromatic subdivision, and we refer to it as $\mathcal{X}(\mathcal{K})$. Whenever the distinction between the geometric and abstract representations of $\mathcal{X}(\mathcal{K})$ is significant, it will be mentioned explicitly.

3.4.2. The nonuniform chromatic subdivision. In this section, we define the *nonuniform chromatic subdivision* and prove that this definition does indeed specify a chromatic subdivision of a given complex. We will give a recursive definition of the *nonuniform chromatic subdivision* which we denote as $\widetilde{\mathcal{X}}^k(\mathcal{K})$, $k \geq 0$. We note that unlike $\mathcal{X}^k(\mathcal{K})$, $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a procedure and not a function, and so $\widetilde{\mathcal{X}}^k(\widetilde{\mathcal{X}}(\mathcal{K})) \neq \widetilde{\mathcal{X}}(\widetilde{\mathcal{X}}^k(\mathcal{K}))$.

DEFINITION 3.30. *Let $\mathcal{K}$ be a pure n-dimensional chromatic complex, where the colors are the numbers in $0, \ldots, n$. A k-level nonuniform chromatic subdivision of a complex $\mathcal{K}$ by $\widetilde{\mathcal{X}}^k(\mathcal{K})$ for $k \geq 0$ is defined as follows.*

If $\dim(\mathcal{K}) = 0$, then for all $k \geq 0$, $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is $\mathcal{K}$ itself. Now suppose $\dim(\mathcal{K}) > 0$. Then $\widetilde{\mathcal{X}}^0(\mathcal{K})$ is $\mathcal{K}$ itself. For $k > 0$, $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is given by the following procedure: Partition the vertexes of $\mathcal{K}$ into two disjoint sets, A and B, where A is nonempty. Let $\mathcal{A}$ and $\mathcal{B}$ be the pure subcomplexes of $\mathcal{K}$ induced (respectively) by the vertexes in A and the vertexes in B. The subdivision $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is the complex consisting of all simplexes in $\mathcal{B}$, all simplexes in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, and all simplexes of the form $S \star T$, where S is a simplex in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T is a simplex in $\mathcal{B}$, and $carrier(S) \star T$ is a simplex in $\mathcal{K}$.

We note that in the above definition, when we say induced subcomplexes $\mathcal{A}$ and $\mathcal{B}$, we mean that a simplex S in $\mathcal{K}$ is in $\mathcal{A}$ if the vertexes spanning S are all in A, and it is in $\mathcal{B}$ if its spanning vertexes are all in B. Since $\mathcal{K}$ is pure, so are $\mathcal{A}$ and $\mathcal{B}$.

Informally speaking, a nonuniform chromatic subdivision of level k is one in which there is some simplex in $\mathcal{K}$ which is subdivided k times but no simplex that is subdivided more than k times. Note that the k-level standard chromatic subdivision is a special case of the k-level nonuniform chromatic subdivision. Hence for all $k \geq 0$, there exists some nonuniform chromatic subdivision of level k.

Our definition of nonuniform chromatic subdivisions is designed to model protocol complexes of the NIIS model. The main difference between IIS and NIIS protocols is that in NIIS, some processes may decide after passing through fewer IS objects than others. This is captured by the recursive definition that splits vertexes into two groups A and B. At any level of the recursion, the vertexes in $\mathcal{A}$ can be thought of as corresponding to processes that continue computing given their current local state, while the vertexes in $\mathcal{B}$ correspond to processes that decide.

To better understand our construction, let us jump slightly ahead of ourselves and consider how the protocol complex of any NIIS protocol with input complex $\mathcal{I}$ in some subset of processes accesses one IS object and is captured by a nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^1(\mathcal{I})$ up to isomorphism.

Consider any vertex $\vec{v}$ in $\mathcal{I}$. It is labeled with $\langle i, v_i \rangle$, where i is a process id and v_i represents an input value to process i. According to the specification of NIIS protocols in section 2.6, process i will (provided it does not fail), upon having received the input v_i, execute either an $inv\_writeread(v)_{i,1}$ action or a $decide(S)_i$ action, depending on whether $\tau(local\_state)$ evaluates to true or not. In this way, the predicate map τ induces a partition of the vertexes of $\mathcal{I}$ into two disjoint sets A and B. We now construct complexes $\mathcal{A}$ and $\mathcal{B}$ as in Definition 3.30; that is, a simplex T in $\mathcal{I}$ is in $\mathcal{A}$ if and only if all its vertexes are in A, and it is in $\mathcal{B}$ if and only if all its vertexes are in B.

The vertexes in A correspond to processes that, based on their input values, execute an $inv\_writeread(v)_{i,1}$ action with the object IS_1. The protocol complex on $\mathcal{A}$ equals $\mathcal{X}(\mathcal{A})$ up to isomorphism. In any execution α of the protocol, some of the participating, nonfailing processes decide on their input values (corresponding to vertexes in B), while some decide on the snapshots they receive from the object IS_1 (corresponding to vertexes in the protocol complex of $\mathcal{A}$). It follows that the protocol complex on $\mathcal{I}$ contains every simplex in $\mathcal{B}$, every simplex in $\mathcal{X}(\mathcal{A})$, and every simplex of the form $S \star T$, where S is in $\mathcal{X}(\mathcal{A})$, T is in $\mathcal{B}$, and $carrier(S) \star T$ is in $\mathcal{I}$.

Note that the structure of the recursion of $\widetilde{\mathcal{X}}^k$ is such that $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ is applied and not $\mathcal{X}(\widetilde{\mathcal{X}}^{k-1}(\mathcal{A}))$. This guarantees that we model a situation in which the subset of processes with nodes A in $\mathcal{K}$ go through the first IS object, and only a subset of these can then go through the next IS object, and so on. It is never the case that a node corresponding to a process that has stopped passing through earlier IS objects is later subdivided.

An example of a level 1 nonuniform chromatic subdivision of a 2-complex $\mathcal{K}$ is given in Figure 17, and an example of a level 2 nonuniform chromatic subdivision of a slightly bigger 2-complex $\mathcal{L}$ is given in Figure 18. Note that in Figure 18 the complex $\mathcal{A}$ for the second level of recursion is isomorphic to the complex $\mathcal{A}$ for the first level of recursion in Figure 17.

An example of a chromatic subdivision that does *not* satisfy Definition 3.30 is given in Figure 19. It is not a nonuniform chromatic subdivision because the vertex
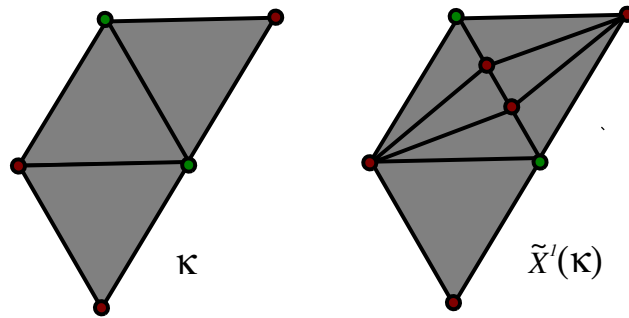
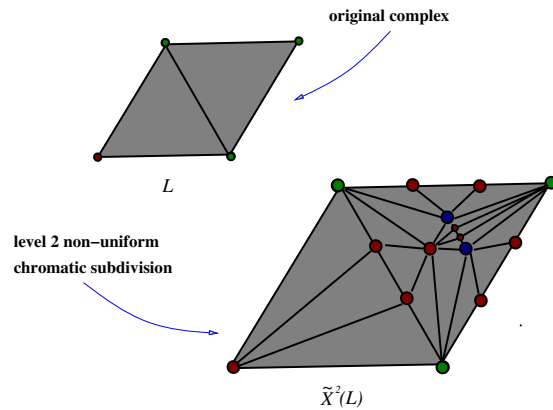FIG. 17. *Example of level 1 nonuniform chromatic subdivision of a 2-complex.*

FIG. 18. *Example of level 2 nonuniform chromatic subdivision of a 2-complex.*

b is part of the $\mathcal{B}$ complex at the first level of recursion (that is, it is not part of the subcomplex that is subdivided further), while in the next level of recursion, the edge between d (which is in the $\mathcal{A}$ complex at the first level of recursion, and hence is to be subdivided further) and b is subdivided, meaning that b is in the $\mathcal{A}$ complex at the second level of recursion, which is clearly impossible, since the carrier of any vertex in the $\mathcal{A}$ complex at the second level must be a simplex in the $\mathcal{A}$ complex at the first level of recursion. Informally, this simply means that, if a vertex is not to be part of the complex to be further subdivided at the first level, it cannot be part of the complex to be further subdivided at the second level.

LEMMA 3.31. *Any nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a chromatic subdivision of $\mathcal{K}$.*

Proof. We first note that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is well defined, since each recursive step lowers the level of subdivision by 1, and $\widetilde{\mathcal{X}}^0(\mathcal{K})$ is defined for all $\mathcal{K}$. We will prove that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a chromatic subdivision by induction on k.

The case where $k = 0$ is trivial, since $\widetilde{\mathcal{X}}^0(\mathcal{K}) = \mathcal{K}$. Now suppose that $k > 0$, and that for $0 \leq l \leq k - 1$, and any complex $\mathcal{K}$, $\widetilde{\mathcal{X}}^l(\mathcal{K})$ is a chromatic subdivision of $\mathcal{K}$.

original complex

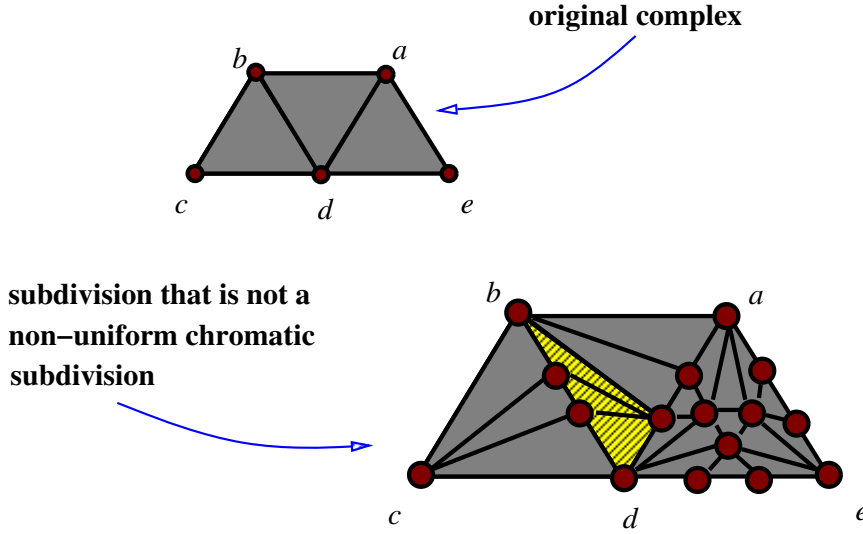**subdivision that is not a non−uniform chromatic subdivision**

FIG. 19. *Example of a subdivision that is* not *a nonuniform chromatic subdivision.*

If $B = \emptyset$, the result follows by induction and by Lemma 3.26. So suppose that B is nonempty.

We first show that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is closed under containment. Let U be a simplex in $\widetilde{\mathcal{X}}^k(\mathcal{K})$, and let V be a face of U. If U is in $\mathcal{B}$, then so is V, since $\mathcal{B}$ is a complex. Hence V is in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. Similarly, if U is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, then so is V, since $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ is a complex by our induction hypothesis. Now suppose $U = S \star T$ for some S in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T in $\mathcal{B}$. Then $S \cap V$ is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, and $T \cap V$ is in $\mathcal{B}$. It follows that $V = (S \cap V) \star (T \cap V)$, where $carrier(S \cap V) \star (T \cap V)$ is a simplex in $\mathcal{K}$. By Definition 3.30, V is in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. It follows that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is closed under containment.

Let U, V be simplexes in $\widetilde{\mathcal{X}}^k(\mathcal{K})$, and let W be their intersection. If both U, V are in $\mathcal{B}$, then so is W, since $\mathcal{B}$ is closed under intersection. Similarly, if both U and V are in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, then so is W, since $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ is closed under intersection. If U is in $\mathcal{B}$ and V is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, or vice versa, then $U \cap V = \emptyset$, and so containment under intersection holds vacuously. We now consider the case where either U or V is not contained in either complex; that is, suppose $U = S \star T$ for some S in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T in $\mathcal{B}$, and $V = X \star Y$ for some X in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, Y in $\mathcal{B}$. Now, $U \cap V = (S \star T) \cap (X \star Y)$, and $(S \star T) \cap (X \star Y) = (S \cap X) \star (T \cap Y)$ [30]. If $S \cap X = \emptyset$ or $T \cap Y = \emptyset$, then since the remaining nonempty intersecting pair of subsets is completely in $\mathcal{B}$ or completely in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, it follows that W is also. So suppose now that S, T, X, Y, and the intersections $S \cap X$ and $Y \cap T$ are all nonempty. Since $carrier(S \cap X)$ is a face of both $carrier(S)$ and $carrier(X)$, it follows that $carrier(S \cap X) \star T$ and $carrier(S \cap X) \star Y$ are simplexes in $\mathcal{K}$, and so is their intersection $carrier(S \cap X) \star (T \cap Y)$, since $\mathcal{K}$ is a complex. It follows that W is in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. This concludes the proof that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a complex. That it is a chromatic complex follows directly from Lemma 3.26.

We now prove that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a chromatic subdivision. Given is a simplex U in $\widetilde{\mathcal{X}}^k(\mathcal{K})$. If U is in $\mathcal{B}$, then U is clearly contained in a simplex in $\mathcal{K}$, namely, itself, and the colors of U are contained in the set of colors of its carrier. If U is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, it follows by induction and Lemma 3.26 that U is contained in some some simplex

carrier(U) in $\mathcal{A}$, and hence in $\mathcal{K}$, and that the colors of U are a subset of the colors of its carrier. Now suppose $U = S \star T$, where *carrier*$(S) \star T$ is in $\mathcal{K}$. Then U is contained in *carrier*$(S) \star T$, and the colors of U are a subset of the colors of *carrier*$(S) \star T$. Now consider any simplex U in $\mathcal{K}$. We can decompose it into two disjoint faces S and T, such that $S \in \mathcal{A}$ and $T \in \mathcal{B}$. The simplex S is subdivided according to $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, which by induction and Lemma 3.26 consists of finitely many simplexes. The simplex T is not subdivided at all. It follows that the subdivision $\widetilde{\mathcal{X}}^k(\mathcal{K})$ subdivides U into finitely many simplexes (those in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{S})) \star T$). This completes the proof that $\widetilde{\mathcal{X}}^k(\mathcal{K})$ is a chromatic subdivision. $\square$

A nonuniform subdivision $\widetilde{\mathcal{X}}^k(\mathcal{K})$ of a complex induces a nonuniform chromatic subdivision of any subcomplex $\mathcal{L}$ of $\mathcal{K}$. The level of the induced subdivision of $\mathcal{L}$ may vary from subcomplex to subcomplex. We slightly abuse our notation to be able to define the effect of $\widetilde{\mathcal{X}}^k(\mathcal{K})$ on the restricted subcomplex $\mathcal{L}$.

DEFINITION 3.32. *Let $\mathcal{K}$ be a chromatic complex, let $\mathcal{L}$ be a subcomplex or simplex of $\mathcal{K}$, and let $\widetilde{\mathcal{X}}^k(\mathcal{K})$ be a nonuniform iterated chromatic subdivision of $\mathcal{K}$. We denote its restriction to simplexes in $\mathcal{L}$ by $\widetilde{\mathcal{X}}^k(\mathcal{L}/\mathcal{K})$. The* level *of subdivision $k_\mathcal{L}$ is the maximal level of subdivision of $\widetilde{\mathcal{X}}^k(\mathcal{L}/\mathcal{K})$.*

It is clear that for any subcomplex or simplex $\mathcal{L}$ of $\mathcal{K}$, $k_\mathcal{L} \leq k$.

4. The asynchronous complexity theorem. The strength and usefulness of the NIIS model of computation comes from the fact that each of its associated protocol complexes has a nice, recursive structure. In fact, it turns out that any protocol complex of NIIS is equal to some nonuniform iterated chromatic subdivision of the input complex, and vice versa. This is the essence of our main theorem, which we state and prove in this section.

The level of subdivision necessary for the existence of a simplicial map from the input to the output complex of a decision task that agrees with the task specification can be interpreted as a topological measure of the task's time complexity. The following definition introduces the concept of *mappability*, which is a useful construct for reasoning about this topological measure.

DEFINITION 4.1. *Given a decision task $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ and a nonnegative integer k, we say that $\widetilde{\mathcal{X}}^k(\mathcal{I})$ is a* mappable *subdivision of the input complex and k is a* mappable *level of subdivision if there exists some chromatic simplicial map μ from $\widetilde{\mathcal{X}}^k(\mathcal{I})$ to $\mathcal{O}$ such that for all T in $\widetilde{\mathcal{X}}^k(\mathcal{I})$, $\mu(T) \in \Gamma(carrier(T))$.*

This definition extends naturally to individual simplexes as the map induces different levels of subdivision on the individual simplexes in accordance with the idea that, in order to solve a decision task, some processes may have to do more computational work than others, and some inputs may require more computation than others. We can now state our main theorem.

THEOREM 4.2 (time complexity). *A decision task $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ has a wait-free solution protocol in the NIIS model with worst case time complexity k_S on inputs $S \in \mathcal{I}$ if and only if there is a mappable nonuniform iterated chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ with level k_S on S.*

Keeping in style with Herlihy and Shavit [24, 25, 26], the theorem simply states that solvability of a decision task $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ in the NIIS model is equivalent to the existence of a chromatic simplicial map μ from some nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ to $\mathcal{O}$ that agrees with the task specification Γ; that is, for all T in $\widetilde{\mathcal{X}}^k(\mathcal{I})$, $\mu(T^m) \in \Gamma(carrier(T))$. The minimum possible level k_S is a lower bound on the worst case time complexity of solving this task with inputs in S in the NIIS model.

As noted in the introduction, the theorem directly implies Proposition 3.1 of [12]. In [12], Borowsky and Gafni provided a simulation of atomic snapshot memory from IIS memory and showed that, based on this simulation, if one is given a constructive proof of an asynchronous computability theorem for the IIS model (which they called Proposition 3.1), it will imply one for the general read-write model. The proof we are about to present provides a constructive proof of computability for the NIIS model, and since IIS is a subset of NIIS, it provides the first known proof of Proposition 3.1 of [12].

Our theorem immediately provides a solution algorithm for a task given the subdivision and simplicial mapping. Simply run the protocol of Figure 6. Since each process can locally store the subdivision and mapping, the termination predicate map τ just needs to test if the *local_state* variable is equal to some node v in the subdivision and if so return $\mu(v)$.

In the remainder of this section, we will give the proof of our asynchronous time complexity theorem. We begin by proving a lemma about the protocol complex of a protocol in the IIS model with only one available IS object.

LEMMA 4.3. *Let $\mathcal{A}$ be an input complex in the IIS model with a single IS object. The corresponding protocol complex is isomorphic to $\mathcal{X}(\mathcal{A})$.*

Proof. We will construct an isomorphism Ψ from the abstract complex $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$ to the abstract complex (vertex scheme) $\mathcal{X}(A)$, as specified by Lemma 3.29. Let $\vec{v} = \langle i, S_i \rangle$ be any vertex in $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$. Then $\Psi(\vec{v}) = \langle i, T_i \rangle$, where T_i is the simplex in $\mathcal{A}$ such that for all j, $S_i[j] = v_j$ if and only if $\langle j, v_j \rangle \in T_i$. Notice that this isomorphism is chromatic; that is, the id of a vertex equals the id of its image under Ψ.

By Lemma 3.29, we must show that a set of vertexes $\vec{v}_0, \ldots, \vec{v}_m$ in $skel^0(\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A}))$, where $m \leq n$, forms a simplex in $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$ if and only if the set of vertexes $\Psi(\vec{v}_0)$, $\ldots, \Psi(\vec{v}_m)$ in $skel^0(\mathcal{X}(\mathcal{A}))$ forms a simplex in $\mathcal{X}(\mathcal{A})$. Suppose without loss of generality that for all i, where $0 \leq i \leq m$, $\vec{v}_i = \langle i, S_i \rangle$, where $S_i \in \vartheta(D_I)$, and D_I is the input data type (that is, the id of the ith vertex is i).

Suppose that the vertexes $\vec{v}_0, \ldots, \vec{v}_m$ do form a simplex V in $\mathcal{P}_{(n,\tau,\delta)}(\mathcal{A})$. This output simplex corresponds to some execution α in the one-shot IS model, with corresponding input simplex U in $\mathcal{A}$. Each vertex in U is labeled with a process id i and an input value $v_i \in D_I$. Notice that $\dim(V) \leq \dim(U)$, since some participating processes may not decide, that is, they may fail (execute a *fail_i* action) before executing a *decide* action.

From Lemma 2.14, we have that, for any vertex $\vec{v}_i = \langle i, S_i \rangle$ in V, $S_i[i] = v_i$. This implies that $\langle i, v_i \rangle$ is in T_i. From Lemma 2.15, we have that, for any two vertexes $\vec{v}_i = \langle i, S_i \rangle$ and $\vec{v}_j = \langle j, S_j \rangle$ in V, either S_i is a prefix of S_j or vice versa. Suppose without loss of generality that S_j is a prefix of S_i. Then for all x, where $0 \leq x \leq n$, if $S_i[x] = \perp$, then $S_j[x] = \perp$, and if $S_j[x] \neq \perp$, then $S_i[x] = S_j[x]$. It follows that if $\langle x, v_x \rangle$ is in T_j, it is also in T_i, and if x is not in $ids(T_i)$, then it is also not in $ids(T_j)$. This implies that T_j is a face of T_i. From Lemma 2.16, it follows that, if $S_i[j] = v_j$, then S_j is a prefix of S_i. This means that, if $\langle j, v_j \rangle$ is in T_i, then T_j is a face of T_i.

Now suppose that the vertexes $\Psi(\vec{v}_0), \ldots, \Psi(\vec{v}_m)$ in $skel^0(\mathcal{X}(\mathcal{A}))$ form a simplex V in $\mathcal{X}(\mathcal{A})$. We will construct an execution α with corresponding output simplex U such that $\Psi(U) = V$. Let $W = carrier(V)$. Partition the set $ids(V)$ into a collection of nonempty *concurrency classes* of process ids, $\mathcal{C}_1, \ldots, \mathcal{C}_k$ for some $k \geq 0$, such that any two process indexes i, j are in the same concurrency class if and only if $T_i = T_j$.

We can define a total order $\prec$ on this collection of concurrency classes as follows. Let $\mathcal{C}_x, \mathcal{C}_y$ be distinct concurrency classes. Then $\mathcal{C}_x \cap \mathcal{C}_y = \emptyset$. Since both classes are nonempty, we can pick an element from each, say, $i \in \mathcal{C}_x$ and $j \in \mathcal{C}_y$. By assumption,

$T_i \neq T_j$. Then by Lemma 3.29, either T_i is a face of T_j or T_j is a face of T_i. In the first case, let $\mathcal{C}_x \prec \mathcal{C}_y$, and in the second case, let $\mathcal{C}_y \prec \mathcal{C}_x$. Thus $\prec$ is a total order of the concurrency classes.

Now use this ordered partition of the participating processes in α to define a second partition $\mathcal{C}'_1, \ldots, \mathcal{C}'_k$ of the set $ids(W)$ as follows. For each concurrency class $\mathcal{C}$ of $ids(V)$, define a concurrency class $\mathcal{C}'$ of $ids(W)$ as follows. $\mathcal{C}'$ is the union of $\mathcal{C}$ and all $i \in ids(W) - ids(V)$ such that $\mathcal{C}$ is the least concurrency class (as determined by $\prec$) such that for all $j \in \mathcal{C}$, $i \in T_j$. Note that this is a partition of all $ids(W)$ since $W = carrier(V)$. This partition gives us a new collection of concurrency classes $\mathcal{C}'_1, \ldots, \mathcal{C}'_k$.

We are now ready to construct α. First position $update_{\mathcal{C}'_i}$ actions in increasing order according to the $\prec$ ordering. For each concurrency class $\mathcal{C}'_x$, position the $inv\_writeread(v)_{i,1}$ actions of all i such that $i \in \mathcal{C}'_x$ immediately before the $update_{\mathcal{C}'_x}$ action (their internal ordering does not matter). Similarly, position the $ret\_writeread(v)_{i,1}$ and $decide(S)_i$ actions of all i such that $i \in \mathcal{C}_x$ and $i \in ids(V)$ immediately after the $update_{\mathcal{C}'_x}$ action, but before the $inv\_writeread(v)_{i,1}$ actions associated with the next concurrency class $\mathcal{C}'_y$. Processes i whose indexes are not in $ids(W)$ do not participate and hence take no steps in α. Processes i whose indexes are in some concurrency class $\mathcal{C}'_x$ but not in $ids(V)$ do not execute a $ret\_writeread(v)_{i,1}$ action; instead, they execute a $fail_i$ action after the $update_{\mathcal{C}'_x}$ action, but before the $inv\_writeread(v)_{i,1}$ actions associated with the next concurrency class $\mathcal{C}'_y$. Recall that earlier concurrency classes could not have included i since by construction $\mathcal{C}'_x$ is the least class including i. By construction, each deciding process i decides S_i in α, as required. The lemma follows. □

We now consider the protocol complex of a protocol in NIIS with time complexity 1 on the input complex $\mathcal{I}$; that is, some processes access a single IS object, while some decide based only on their own inputs. We will show that, if δ is trivial, which we denote by $\delta = 1$, then this protocol complex is indeed a nonuniform chromatic subdivision.

LEMMA 4.4. *For all $k \geq 0$, the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ of any protocol in the NIIS model, with time complexity k on inputs in $\mathcal{I}$, is equal to some nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ up to isomorphism.*

Proof. We use induction on the time complexity k. The result holds for $k = 0$ trivially. Now suppose $k > 0$ and that the result holds for $1, \ldots, k - 1$. Consider the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ of any protocol in the NIIS model with time complexity k on inputs in $\mathcal{I}$.

Any vertex $\vec{v}$ in $\mathcal{I}$ is labeled with $\langle i, v_i \rangle$, where i is a process id and v_i represents an input value to process i. According to the specification of NIIS protocols in section 2.6, any nonfailing process i will, upon having received the input v_i, execute either an $inv\_writeread(v)_{i,1}$ action or a $decide(S)_i$ action, depending on whether $\tau(local\_state)$ evaluates to true or not. In this way, the predicate map τ induces a partition of the vertexes of $\mathcal{I}$ into two disjoint sets A and B. Since the time complexity of $\mathcal{P}_{(n,\tau,1)}$ on inputs in $\mathcal{I}$ is k, the set A must be nonempty. We now construct complexes $\mathcal{A}$ and $\mathcal{B}$ as in Definition 3.30; that is, a simplex T in $\mathcal{I}$ is in $\mathcal{A}$ if and only if all its vertexes are in A, and it is in $\mathcal{B}$ if and only if all its vertexes are in B.

The vertexes in A correspond to processes that, based on their input values, execute an $inv\_writeread(v)_{i,1}$ action with the object IS_1. By Lemma 4.3, the output protocol complex on inputs in $\mathcal{A}$ after the first IS access equals $\mathcal{X}(\mathcal{A})$ up to isomorphism. The final protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{A})$ is given by applying $\mathcal{X}(\mathcal{A})$ as an input complex to the protocol. Since the complexity of the protocol on inputs in $\mathcal{I}$, and hence

on inputs in $\mathcal{A}$, is k, the complexity of the protocol on inputs in $\mathcal{X}(\mathcal{A})$ must be $k-1$. It follows by induction that the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{A})$ equals some nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$ of $\mathcal{X}(\mathcal{A})$ up to isomorphism. A simplex U is in $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ if and only if it corresponds to a valid set of outputs of an execution α of the protocol. In any execution α of the protocol, some of the participating, nonfailing processes decide on their input values (corresponding to vertexes in B), while some decide on the returned snapshots they receive from some IS object. It follows that $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ contains any simplex in $\mathcal{B}$, any simplex in $\mathcal{P}_{(n,\tau,1)}(\mathcal{A}) = \widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, and any simplex of the form $S \star T$, where S is in $\widetilde{\mathcal{X}}^{k-1}(\mathcal{X}(\mathcal{A}))$, T is in $\mathcal{B}$, and $carrier(S) \star T$ is in $\mathcal{I}$. It follows from Definition 3.30 that the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ equals some nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ of $\mathcal{I}$ up to isomorphism. $\quad\square$

We must also prove that, for any mappable nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ of an input complex $\mathcal{I}$, there is a matching protocol $\mathcal{P}_{(n,\tau,1)}$ in the NIIS model.

LEMMA 4.5. *For any mappable nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ of an input complex $\mathcal{I}$, there is a matching protocol $\mathcal{P}_{(n,\tau,1)}$ in the NIIS model such that the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I}) = \widetilde{\mathcal{X}}^k(\mathcal{I})$ up to isomorphism.*

Proof. Given is a vertex $\vec{v}$ in $\mathcal{I}$. The definition of the subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ induces a sequence of nonuniform chromatic subdivisions $\mathcal{I}, \widetilde{\mathcal{X}}^1(\mathcal{I}), \dots, \widetilde{\mathcal{X}}^k(\mathcal{I})$ and corresponding sequences $\mathcal{A}_0, \dots, \mathcal{A}_{k-1}$ and $\mathcal{B}_0, \dots, \mathcal{B}_{k-1}$ of complexes, the former sequence specifying the subcomplex to be subdivided further at each level of recursion.

In order to construct a protocol for $n+1$ processes, we must specify the function $\tau : \bigcup_{l=0}^k \vartheta^l(D) \to \{\texttt{true}, \texttt{false}\}$ and the decision map $\delta : \bigcup_{l=0}^k \vartheta^l(D) \to D_O$. We specify τ to be $\texttt{true}$ for all values v such that there is a vertex $\vec{v}$ in one of the complexes $\mathcal{A}_0, \dots, \mathcal{A}_{k-1}$ with $val(\vec{v}) = v$. For all other values v, τ evaluates to $\texttt{false}$. This definition is well formed, since for all p, where $0 \le p \le k$, it follows from Definitions 3.28 and 3.30 that there are no two vertexes in $\widetilde{\mathcal{X}}^p(\mathcal{I})$ with the same process-value label pair, and for all p, q, where $0 \le p, q \le k$ and $p \ne q$, $\mathcal{B}_p$ and $\mathcal{B}_q$ have no vertexes with common labels (process id *and* value label). This concludes the proof. $\quad\square$

We now give the proof of Theorem 4.2.

Proof of Theorem 4.2. Let $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ be a decision task. Lemma 4.4 states that any protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$, with worst case complexity k_S on input S, corresponds to a nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ with level k_S on S. Suppose now the decision map δ is not trivial. Then, if $\mathcal{P}_{(n,\tau,\delta)}$ solves $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$, $\mu = \delta$ is a simplicial map from $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ to $\mathcal{O}$ that is in correspondence with Γ, so $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ is mappable.

Lemma 4.5 states that any mappable nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ with level k_S on S is equal to the protocol complex $\mathcal{P}_{(n,\tau,1)}(\mathcal{I})$ (where δ is trivial) of a protocol in the NIIS model with worst case complexity k_S on input S. If there is a simplicial map μ from $\widetilde{\mathcal{X}}^k(\mathcal{I}^n)$ to $\mathcal{O}$ that is consistent with Γ, then by setting $\delta = \mu$, we have a protocol $\mathcal{P}_{(n,\tau,\delta)}$ solving $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ with complexity k_S on input S. The theorem follows. $\quad\square$

5. Approximate Agreement. As an application of Theorem 4.2, we analyze the well-known *Approximate Agreement* [15] task, defined as follows: each process $i \in \{0, \dots, n\}$ has an input x_i taken from some finite subset of the reals and chooses a unique output y_i such that, for some predetermined $\epsilon \ge 0$, (1) $\max_i y_i - \min_i y_i < \epsilon$, and (2) for all i, $y_i \in [\min_i x_i, \max_i x_i]$.

This problem, which at first glance may seem similar to Consensus, is in fact quite different and is solvable in the read-write memory model. (If ϵ were 0 this problem

would be equivalent to Consensus and hence not solvable.) Aspnes and Herlihy [2]
proved a lower bound on Approximate Agreement in the read-write memory model
that implies a worst case time complexity of $\lceil \log_3 \frac{\max_i x_i - \min_i x_i}{\epsilon} \rceil$ and an upper bound
of $\lceil \log_2 \frac{\max_i x_i - \min_i x_i}{\epsilon} \rceil$ in the NIIS model. We will show that this $\log_2$ versus $\log_3$
gap is not simply a technical fluke.

DEFINITION 5.1. *Let V be some finite subsequence of values from $\mathbb{R}$, at most ϵ
apart from its successor. The finite $n + 1$-process* Approximate Agreement *task is the
tuple $\mathcal{D} = \langle I, O, \gamma \rangle$:*

- $I = \{[x_0, \ldots, x_n] \mid x_i \in V \cup \{\perp\}\}$.
- $O = \{[y_0, \ldots, y_n] \mid y_i \in V \cup \{\perp\}, (y_i, y_j \neq \perp) \Rightarrow |y_i - y_j| \leq \epsilon\}$.
- $\gamma = \{(\vec{I}, \vec{O}) \mid \vec{O}[i] \in [\min_i \vec{I}[i], \max_i \vec{I}[i]] \cup \{\perp\}\}$.

Define an input vector $\vec{I}$ to be *nontrivial* if the $\max_i x_i$ and $\min_i x_i$ are at least ϵ
apart and each belongs to at least one other disjoint input vector. We can now state
the complexity bounds for the Approximate Agreement problem.

THEOREM 5.2. *Given $\epsilon > 0$, there is a protocol $\mathcal{P}_{(n,\tau,\delta)}$ solving* Approximate
Agreement *for any nontrivial input vector $\vec{I}$ with complexity $\lceil \log_d \frac{\max_i \vec{I}[i] - \min_i \vec{I}[i]}{\epsilon} \rceil$,
where $d = 3$ if the size of the participating set of $\vec{I}$ is 2, and $d = 2$ if the size of the
participating set of $\vec{I}$ is 3 or more. Moreover, this protocol is optimal for $\vec{I}$.*

We note that in many cases, for trivial input vectors one can "statically" predefine
the outputs for each input value so that no access to an *IS* object is necessary.

Our proof structure will be as follows. The upper bound will follow by showing
a subdivision and simplicial map and then applying Theorem 4.2. The lower bound
proof will follow from a geometric observation regarding the structure of any NIIS
subdivided complex for approximate agreement.

We first restate the description of the Approximate Agreement task using our
topological framework.

- $\mathcal{I}$ is the closure under containment of the collection of all simplexes of the
  form $(\langle 0, x_0 \rangle, \ldots, \langle n, x_n \rangle)$.
- $\mathcal{O}$ is the closure under containment of the collection of all simplexes of the
  form $(\langle 0, y_0 \rangle, \ldots, \langle n, y_n \rangle)$, where for all $i, j, y_i \in V$ and $|y_i - y_j| \leq \epsilon$.
- $\Gamma = \{(S, T) \mid vals(T) \subseteq [\min vals(S), \max vals(S)]\}$.

Note that the size of the participating set for the input vector corresponding to
a simplex S in $\mathcal{I}$ equals $\dim(S) + 1$.

To understand our proof approach, consider Figure 20, which shows the subdivi-
sions induced by a three process protocol. In [2], the lower bound for any $n+1$-process
algorithm is derived from a "bad" execution in which only the two processes P and
Q with inputs farthest apart participate. Cast in our model, processes P and Q have
inputs p and q (the corners of the input simplex in Figure 20). Because there are
other simplexes adjacent to the input 1-simplex $(\langle P, p \rangle, \langle Q, q \rangle)$ that share only the
input value p and, respectively, q, μ must map $\langle P, p \rangle$ to output value p and $\langle Q, q \rangle$ to
output value q. An execution in the NIIS model corresponds to a sequence of chro-
matic subdivisions of the edge $(\langle P, p \rangle, \langle Q, q \rangle)$ (a path of 1-simplexes) from which there
is a simplicial map to a path in the output complex. In the end of the execution, the
vertexes of each 1-simplex along this edge must be mapped to an output 1-simplex
with values less than ϵ apart. Each subdivision, corresponding to an NIIS execution
step, introduces two new vertexes and splits the edge $(\langle P, p \rangle, \langle Q, q \rangle)$ in three. This
implies that reaching a distance of ϵ or less along the path of simplexes in the sub-
division of $(\langle P, p \rangle, \langle Q, q \rangle)$ requires at least $\log_3(\text{distance}(p, q)/\epsilon)$. This is the bound
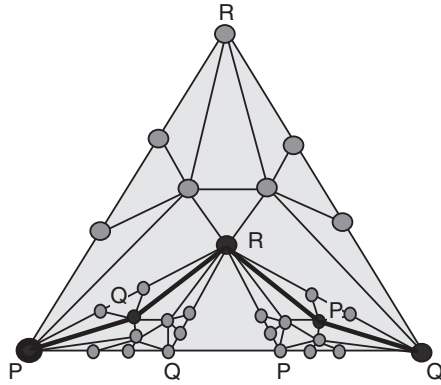of [2].

FIG. 20. *Simplex subdivided by an Approximate Agreement protocol.*

However, if one considers executions in which three processes participate, this proof does not work. Consider the first round of subdivision of the simplex for three processors. There is a path (a sequence of adjoining 1-simplexes) between the two endpoints P and Q, along which only a single vertex is introduced by the subdivision. This is the path of length two through the central vertex marked R in Figure 20 (it is not marked since its simplexes are further subdivided in the figure). So the maximum distance is cut by at most a half in the first subdivision. In the next step of subdivision, even though each of the original 1-simplexes (P, R) and (R, Q) of $\widetilde{\mathcal{X}}(S)$ can be subdivided into three simplexes, there is still a path (highlighted in the figure) from P to Q along which only a single node was added connecting P and R (and, respectively, R and Q). In general, after k subdivisions, there is always a path that was divided only 2^k times—hence the lower bound of $\log_2(\text{distance}(p, q)/\epsilon)$. Our upper bounds follow directly from Theorem 4.2 by specifying the proper subdivision and map. The thing to note about the proof we will present for Theorem 5.2 is that it will not involve any mention of the actual executions; all we need to do is argue about the topology of the inputs and outputs and then apply Theorem 4.2.

Proof. Theorem 5.2 states that, given $\epsilon > 0$, there is a protocol $\mathcal{P}_{(n,\tau,\delta)}$ solving Approximate Agreement with complexity $\left\lceil \log_d \frac{\max vals(S) - \min vals(S)}{\epsilon} \right\rceil$ on any input simplex S, where $d = 3$ if $\dim(S) = 1$ and $d = 2$ if $\dim(S) \geq 1$. Moreover, this protocol is optimal on each input simplex S.

We first establish the lower bound. Let $\mathcal{P}_{(n,\tau,\delta)}$ be a protocol that solves Approximate Agreement with worst case complexity k_S on S, where S is any simplex of dimension $n \geq \dim(S) > 0$. Let $D(S) = \max_{\vec{v}, \vec{u} \in S}(val(\vec{v}) - val(\vec{u}))$ and let $D(\widetilde{\mathcal{X}}^k(\mathcal{I}))$ equal $\max_{S \in \widetilde{\mathcal{X}}^k(\mathcal{I})} D(S)$. Then Theorem 4.2 states that there is some mappable nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$, with level k_S on S. We will show that $k_S \geq \left\lceil \log_d \frac{D(S)}{\epsilon} \right\rceil$. The proof uses the following lemma.

LEMMA 5.3. *Let $l \leq k$. Label the vertexes of $\widetilde{\mathcal{X}}^l(S/\mathcal{I})$ with real numbers in a way that agrees with the initial value labeling of S, and let l_S be the level of $\widetilde{\mathcal{X}}^l(S/\mathcal{I})$. Then*

$$D(\widetilde{\mathcal{X}}^l(S/\mathcal{I})) \geq \frac{D(S)}{d^{l_S}}.$$

Proof. Suppose without loss of generality that $l = l_S$. We first give the proof for the case of two participating processes and $d = 3$. By definition of $D(S)$, there

is a 1-simplex $U = (\vec{u_0}, \vec{u_1})$ in $\mathcal{S}$ such that $D(U) = D(S)$. The complex $\widetilde{\mathcal{X}}^l(U)$ contains at most 3^l 1-simplexes, denoted $U_1, \ldots, U_M$, where $M \leq 3^l$. These form a continuous path from $\vec{u}_0$ to $\vec{u}_1$, the endpoints of which are labeled with $val(\vec{u}_0)$ and $val(\vec{u}_1)$, respectively. So the best we can do is cut $D(U)$ into 3^l pieces. The triangle inequality tells us that $D(U) \leq \sum_{i=1}^{M} D(U_i) \leq M \max_i D(U_i) \leq 3^l \max_i D(U_i)$. Hence $\max_i D(U_i) \geq D(U)/3^l = D(S)/3^l$. The lemma follows, since $\max_i D(U_i) \leq D(\widetilde{\mathcal{X}}^l(S/\mathcal{I}))$.

We now prove the case where the size of the participating set is greater than 2 (and hence $\dim(S)$ is greater than 1) and $d = 2$. We argue by induction on l. The case $l = 0$ is trivial. Now suppose the claim is true for $l - 1$. By definition of $D(\widetilde{\mathcal{X}}^{l-1}(S/\mathcal{I}))$, there is a 1-simplex $U = (\vec{u_0}, \vec{u_1})$ in $\widetilde{\mathcal{X}}^{l-1}(S/\mathcal{I})$ such that $D(U) = D(\widetilde{\mathcal{X}}^{l-1}(S/\mathcal{I}))$. U is a face of some 2-simplex $U' = (\vec{u_0}, \vec{u_1}, \vec{u_2})$. Suppose first that the next level of nonuniform chromatic subdivision does not subdivide U completely. Then there is some 1-simplex T in the level l nonuniform subdivision of U' with $D(T) \geq D(U)/2$. Since $D(U) = D(\widetilde{\mathcal{X}}^{l-1}(S/\mathcal{I}))$ and $D(T) \leq D(\widetilde{\mathcal{X}}^l(S/\mathcal{I}))$, the lemma follows by induction. Suppose instead that the next level of subdivision does subdivide U' completely. Then the level l subdivision has an internal vertex $\vec{m}_2$, colored with $id(\vec{u}_2)$, and two neighboring 1-simplexes $T_0 = (\vec{u}_0, \vec{m}_2)$ and $T_1 = (\vec{m}_2, \vec{u}_1)$. The triangle inequality then tells us that $D(U) \leq D(T_0) + D(T_1) \leq 2 \max_i D(T_i)$, where $i \in \{0, 1\}$. It follows that $D(\widetilde{\mathcal{X}}^l(S/\mathcal{I})) \geq D(\widetilde{\mathcal{X}}^{l-1}(S/\mathcal{I}))/2$. The lemma follows by induction. $\quad\square$

Suppose now that there exists a chromatic simplicial map $\mu : \widetilde{\mathcal{X}}^k(\mathcal{I}) \to \mathcal{O}$ such that, for all simplexes T in $\widetilde{\mathcal{X}}^k(\mathcal{I})$, $\mu(T) \in \Gamma(carrier(T))$. We can associate this map with a labeling of the vertexes in $\widetilde{\mathcal{X}}^k(\mathcal{I})$ as follows. Label each vertex $\vec{v}$ in $\widetilde{\mathcal{X}}^k(\mathcal{I})$ with $val(\mu(\vec{v}))$. This labeling agrees with the input value labeling of $\mathcal{I}$, since for any vertex $\vec{v}$, the task specification requires that for any simplex S_0 that contains $\vec{v}$, it must be the case that $\mu(\vec{v})$ is in the range of S_0. Based on the nontriviality assumption, choose two neighboring simplexes S_0 and S_1 containing $\vec{v}$ such that the intersection of the ranges of S_0 and S_1 is $val(\vec{v})$. It follows that $\mu(\vec{v}) = val(\vec{v})$. Now let T be any simplex in $\widetilde{\mathcal{X}}^k(\mathcal{I})$. By definition of μ, $\mu(T)$ is a simplex in $\mathcal{O}$, and hence $D(\mu(T)) < \epsilon$. It follows that $D(T) = D(\mu(T)) < \epsilon$ and hence that $D(\widetilde{\mathcal{X}}^k(\mathcal{I})) < \epsilon$. Clearly, for any input simplex S, it follows that the labels on $\widetilde{\mathcal{X}}^k(S/\mathcal{I})$ have range less than ϵ. The previous lemma then states that $\epsilon > D(\widetilde{\mathcal{X}}^k(S/\mathcal{I})) \geq \frac{D(S)}{d^{k_S}}$. We conclude that

$$k_S \geq \left\lceil \log_d \frac{D(S)}{\epsilon} \right\rceil.$$

To prove the upper bound, we construct a mappable nonuniform chromatic subdivision $\widetilde{\mathcal{X}}^k(\mathcal{I})$ of the input complex with level $k_S = \left\lceil \log_d \frac{D(S)}{\epsilon} \right\rceil$ on each input simplex S, according to Definition 3.30. As argued above, the requirement that the subdivision be mappable is equivalent to saying that there is a vertex labeling of $\widetilde{\mathcal{X}}^k(\mathcal{I})$ that agrees with the initial value labeling of $\mathcal{I}$ with the additional property that $D(\widetilde{\mathcal{X}}^k(\mathcal{I})) < \epsilon$.

Apply $\widetilde{\mathcal{X}}^k$ to $\mathcal{I}$ where for every input n-simplex $S \in \mathcal{I}$, and construct $\widetilde{\mathcal{X}}^k(S/\mathcal{I})$ by repeatedly applying the procedure $\widetilde{\mathcal{X}}^k$ (as specified by Definition 3.30) for each level of subdivision $l \leq k$. For each level, split the vertexes into two sets so that a vertex $\vec{v}$ is in A if there is another adjacent vertex $\vec{u}$ such that $val(\vec{v}) - val(\vec{u}) > \epsilon$; otherwise it is in B. Before applying the next level of subdivision to $\mathcal{X}(A)$, we relabel all new vertexes in $\mathcal{X}(A)$ (those not in $skel^0(A)$) as follows: If the dimension of A is 1,

label the new vertexes in $\mathcal{X}(\mathcal{A})$ with $(2 \min val(\mathcal{A}) + \max val(\mathcal{A}))/3$ and $(\min val(\mathcal{A}) + 2 \max val(\mathcal{A}))/3$, respectively. This cuts the distance between the vertexes with values apart in 3. Otherwise, label the new vertexes with $(\min val(\mathcal{A}) + \max val(\mathcal{A}))/2$. This cuts the distance between the values furthest apart in 2.

It is clear from this construction that, at each level of recursion, for all simplexes S in $\mathcal{I}$ we have that, if $D(\widetilde{\mathcal{X}}^l(S)) > \epsilon$, then either $D(\widetilde{\mathcal{X}}^{l+1}(S)) = D(\widetilde{\mathcal{X}}^l(S))/d$ or $D(\widetilde{\mathcal{X}}^{l+1}(S)) < \epsilon$. It follows that the level k_S of $\widetilde{\mathcal{X}}^k(\mathcal{I})$ on S is $\left\lceil \log_d \frac{D(S)}{\epsilon} \right\rceil$, where $d = 3$ if $\dim(S) = 1$, and $d = 2$ if $\dim(S) > 1$. We conclude from Theorem 4.2 that there is a wait-free protocol that solves Approximate Agreement with worst case time complexity $\left\lceil \log_d \frac{D(S)}{\epsilon} \right\rceil$ on input S, where $d = 3$ for two participating processes and $d = 2$ for three or more.   □

6. Conclusion and directions for further research. This paper extended the topological framework of Herlihy and Shavit [24, 25, 26] to obtain a complete characterization of the complexity of solving decision tasks in the NIIS model, a generalization of Borowsky and Gafni's IIS model [12]. The main difference between the proof of Theorem 4.2 and Herlihy and Shavit's proof of their asynchronous computability theorem is that our proof rests on the ability to explicitly construct a protocol complex for the NIIS model and to show that this complex is indeed equal to a nonuniform chromatic subdivision. Since nonuniform chromatic subdivisions have a recursive structure, they are well suited for arguing about complexity; the level of recursion of a mappable nonuniform chromatic subdivision of a task's input complex *is* the complexity of the corresponding wait-free NIIS solution protocol.

We have applied Theorem 4.2 to tighten the upper and lower bounds on solving the *Approximate Agreement* task implied by the work of Aspnes and Herlihy [2]. The intuition behind this result as well as its formal proof are based on simple, geometric, and topological arguments about the level of nonuniform chromatic subdivision that is necessary and sufficient for mappability. We believe this is an excellent example of how Theorem 4.2 exposes subtle properties of protocols in asynchronous shared memory systems and how it allows us to reason formally about them without having to argue directly about concurrent executions.

We believe it is possible to extend our existing topological framework to develop a characterization of *work complexity*, the total number of steps taken by all processors in a computation. As was the case for time complexity, a mappable nonuniform chromatic subdivision of an input complex does contain the information necessary to describe work complexity, and the question is really how to quantify and measure it using a simple topological invariant.

Another possible direction is to try to extend the framework to other models of computation, such as the atomic snapshot model, the single-writer multireader model, or even the multiwriter multireader model. Our choice of the NIIS model was motivated by the fact that its protocol complex is highly structured and corresponds to a nonuniform chromatic subdivision, as the proof of Theorem 4.2 shows. Other, less restricted, models, such as atomic snapshots, do not have this property, and so in order to prove a result similar to Theorem 4.2 in any of these models, one would need to identify some invariant, recursive substructure that one can model topologically with reasonable ease.

An alternative approach would be to use simulation techniques to relate the NIIS model to other models of computation, thereby obtaining an indirect characterization of the complexity of solving decision tasks in these models. Currently, however, the best known wait-free simulation of a single IS object using atomic snapshots re-

quires $O(N)$ accesses to shared memory by each process, where N is the number of processes. There is thus an important open problem in finding an *optimal*, wait-free implementation of NIIS using atomic snapshot, and vice versa.

Acknowledgments. We wish to thank Nancy Lynch for her comments on initial drafts of this paper. We also thank the anonymous referees for their careful reading of the manuscript and for providing an abundance of constructive comments.

REFERENCES

[1] Y. AFEK, H. ATTIYA, D. DOLEV, E. GAFNI, M. MERRITT, AND N. SHAVIT, *Atomic snapshots of shared memory*, J. ACM, 40 (1993), pp. 873–890.

[2] J. ASPNES AND M. P. HERLIHY, *Wait-Free Data Structures in the Asynchronous Pram Model*, manuscript, Brown University, Providence, RI, 1996.

[3] H. ATTIYA AND S. RAJSBAUM, *A Combinatorial Topology Framework for Wait-Free Computability*, preprint, The Technion, Haifa, Israel, 1995.

[4] H. ATTIYA, A. BAR-NOY, AND D. DOLEV, *Sharing memory robustly in message-passing systems*, J. ACM, 42 (1995), pp. 124–142.

[5] H. ATTIYA, A. BAR-NOY, D. DOLEV, D. PELEG, AND R. REISCHUK, *Renaming in an asynchronous environment*, J. ACM, 37 (1990), pp. 524–548.

[6] H. ATTIYA, N. LYNCH, AND N. SHAVIT, *Are wait-free algorithms fast?*, J. ACM, 41 (1994), pp. 725–763.

[7] O. BIRAN, S. MORAN, AND S. ZAKS, *A combinatorial characterization of the distributed 1-solvable tasks*, J. Algorithms, 11 (1990), pp. 420–440.

[8] E. BOROWSKY, *Capturing the Power of Resiliency and Set Consensus in Distributed Systems*, Technical report, University of California Los Angeles, Los Angeles, CA, 1995.

[9] E. BOROWSKY, E. GAFNI, N. LYNCH, AND S. RAJSBAUM, *The BG distributed simulation algorithm*, Distrib. Comput., 14 (2001), pp. 127–146.

[10] E. BOROWSKY AND E. GAFNI, *Generalized FLP impossibility result for t-resilient asynchronous computation*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 91–100.

[11] E. BOROWSKY AND E. GAFNI, *Immediate atomic snapshots and fast renaming*, in Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1993, pp. 41–51.

[12] E. BOROWSKY AND E. GAFNI, *A simple algorithmically reasoned characterization of wait-free computation (extended abstract)*, in Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1997, pp. 189–198.

[13] S. CHAUDHURI, *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, ACM, New York, 1990, pp. 311–234.

[14] S. CHAUDHURI, M. HERLIHY, N. A. LYNCH, AND M. R. TUTTLE, *Tight bounds for k-set agreement*, J. ACM, 47 (2000), pp. 912–943.

[15] D. DOLEV, N. A. LYNCH, S. S. PINTER, E. W. STARK, AND W. E. WEIHL, *Reaching approximate agreement in the presence of faults*, J. ACM, 33 (1986), pp. 499–516.

[16] A. D. FEKETE, *Asymptotically optimal algorithms for approximate agreement*, in Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1986, pp. 73–87.

[17] M. FISCHER, N. A. LYNCH, AND M. S. PATERSON, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.

[18] E. GAFNI AND E. KOUTSOUPIAS, *Three-processor tasks are undecidable*, SIAM J. Comput., 28 (1999), pp. 970–983.

[19] J. HAVLICEK, *Computable obstructions to wait-free computability*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97), IEEE Computer Society, Los Alamitos, CA, 1997, pp. 80–89.

[20] M. HERLIHY AND S. RAJSBAUM, *Set consensus using arbitrary objects (preliminary version)*, in Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1994, pp. 324–333.

[21] M. HERLIHY AND S. RAJSBAUM, *The decidability of distributed decision tasks (extended abstract)*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 589–598.

[22] M. HERLIHY AND S. RAJSBAUM, *Algebraic spans*, Math. Structures Comput. Sci., 10 (2000), pp. 549–573.

[23] M. HERLIHY, S. RAJSBAUM, AND M. R. TUTTLE, *Unifying synchronous and asynchronous message-passing models*, in Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1998, pp. 133–142.

[24] M. HERLIHY AND N. SHAVIT, *The asynchronous computability theorem for t-resilient tasks*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 111–120.

[25] 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, ACM, New York, 1994, pp. 243–252.

[26] M. HERLIHY AND N. SHAVIT, *The topological structure of asynchronous computability*, J. ACM, 46 (1999), pp. 858–923.

[27] M. P. HERLIHY, *Wait-free synchronization*, ACM Trans. Programming Languages and Systems, 13 (1991), pp. 123–149.

[28] N. A. LYNCH AND M. R. TUTTLE, *An introduction to input/output automata*, CWI Quarterly, 2 (1989), pp. 219–246.

[29] N. A. LYNCH, *Distributed Algorithms*, Morgan–Kaufman, San Francisco, CA, 1996.

[30] J. R. MUNKRES, *Elements of Algebraic Topology*, Addison–Wesley, Reading, MA, 1984.

[31] G. NEIGER, *Set-linearizability*, in Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1994, p. 396.

[32] M. SAKS AND F. ZAHAROGLOU, *Wait-free k-set agreement is impossible: The topology of public knowledge*, SIAM J. Comput., 29 (2000), pp. 1449–1483.

[33] E. SCHENK, *Computability and Complexity Results for Agreement Problems in Shared Memory Distributed Systems*, Technical report, University of Toronto, Toronto, Canada, 1996.

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