Consensus as a Form of Resiliency

Elizabeth Borowsky (borowsky@cs.ucla.edu)
Eli Gafni (eli@cs.ucla.edu)

UCLA Computer Science Department
Los Angeles, CA 90024

April 26, 1995

Abstract

In the traditionally investigated t-resilient model, it is assumed that in the worst case at most t of the n processors in the system may fail-stop. When significantly less than n processors participate in a computation all t faulty processors may be among the participating processors, in the worst case. However, this does not sit well with the underlying assumption that there is a fixed probability of failure for each processor. Thus a natural model to consider is that a certain fraction of the participating processors may fail. What then is the set of tasks solvable in such a model?

This paper answers this question as a corollary to its main technical contribution, which is a statement and a proof of the necessary and sufficient conditions for solvability of tasks in models with traditional resiliency and active-resiliency conditions in which processors have access to set-consensus objects and read-write registers. It then shows that a remarkable corollary of the conditions is that tasks solvable with less than half of the active processors faulty are exactly the tasks solvable wait-free with two process consensus objects. In general, it shows that with any set S of consensus object types one can associate a function $r(i)$ such that the read-write model in which if i processors are active then at most $r(i)$ are faulty is equivalent to the wait-free model using objects from S. Additionally, it closes the gap in knowledge that exists between the one-resiliency solvability conditions of [1] and the wait-free solvability conditions of [2, 3].

1 Introduction

Since 1985, when Fisher, Lynch, and Patterson [4] proved that the consensus task is not solvable even in the face of a single failure, there has been an intense focus on understanding which problems are solvable and which unsolvable in a given model of a distributed system. A programmer of a given distributed system may use such knowledge to understand the limits of the synchronization power of that system and under what resiliency assumption a given task may be solved. In addition, such understanding may influence the design of the system and the programming language with respect to the question of which primitives to provide.

In their pioneering work of 1988, Biran, Moran, and Zaks [1] made a connection between task solvability and graph theory. They looked at the traditional model of one-resiliency, in which only one processor out of a given number n may fail. They completely characterized the conditions for solvability in this model, basing their characterization on the connectivity of a graph derived from the task input/output relation.

In 1990, Chaudhuri [5] proposed the task of k-set-consensus, conjecturing that it is solvable in spite of $k-1$, but not k, fail-stop failures. In 1993 three teams proved Chaudhuri's conjecture correct. Herlihy and Shavit [2], Borowsky and Gafni [6], and Saks and Zaharoglou [7] all showed that $k+1$ processors working wait-free (i.e., k possible faults) in read-write memory can not solve k-set-consensus. All of these teams, as well as Chaudhuri [5], used algebraic-topology techniques. Herlihy and Shavit [2, 3] identified the general scheme of characterizing a model via algebraic topology and characterized the read-write waitfree model.

Finding the necessity part of a characterization involves capturing properties (protocol-properties) that will be present in any complex that corresponds to a protocol in the model. To prove these properties sufficient one produces a "universal" protocol in the model, that can converge to a simplex on any complex that satisfies the protocol-properties.

In this paper, we characterize any t-resilient model that allows access to set-consensus objects. Furthermore, we introduce a new model of resiliency, called *active-resiliency*, and characterize it too. Why is this model interesting? In the t-resilient model, at most t of the processors in the system may fail-stop. Thus even a processor that takes no steps (i.e. sleeping) through-out the execution, is considered faulty. However, in many applications, a processor that sleeps throughout the execution is not considered faulty, but rather a no-show. If the protocol is to resolve contention for resources, the fact that a processor sleeps throughout the execution may mean that the processor does not need the resource rather than that it failed. Thus, exclusion protocols consider a sleeping processor (in the remainder section) simply as not showing-up rather than as a failure. It is thus natural and interesting to consider models in which only an active (awake and not halted) processor that has stopped taking steps can be considered faulty. While in the t-resilient model we have a protocol over a constant number, n, of processors, in the active-resiliency model the number of processors changes over the course of the execution. In such a model, it makes sense to talk about at most $r(i)$ of the currently i active processors failing. In fact, t-resiliency may be considered as an active-resiliency model in which $r(i) = i-(n-t), i \geq n-t$, with the variation that no processor may terminate before $n-t$ processors wake up.

What is the characterization of any of the resiliency models, the traditional or the active? What is it when we add set-consensus objects to the models? The protocol-property of a complex that corresponds to a protocol in a read-write wait-free model is that the complex is a subdivided simplex. In a subdivided simplex, a sphere has a fill-in that makes it a ball inside the smallest face on which the sphere resides. What if we consider complexes for which spheres might not have a fill-in in the smallest face s^i of dimension i they reside in but may have fill-in when considered as part of larger face s^j of a given dimension j that contains it.

This is the case with either the active or t-resiliency model employing set-consensus. For such models, a sphere that resides in a complex corresponding to having only a few processors active may not have a fill-in inside that complex, yet when this sphere is considered inside a large enough complex (depending on the particular parameters of the model) that is the result of more processors becoming active, the sphere may have a fill-in. This was also observed in [8]. Indeed, it turns out that this is a characterization. The growth in the number of processors that is required in order to fill-in spheres of different dimensions (in any protocol-complex in the model) completely specifies whether or not a given task is solvable in a given model of resiliency employing consensus objects. The greater the growth, the more powerful the model. In a model with universal objects (e.g., read-modify-write) or completely resilient with no failure, the growth is infinite. No sphere may have a fill-in but the trivial sphere of a single vertex, no matter how many active processors are added. In read-write wait-free model no increase in the number of processors is required to have a fill-in.

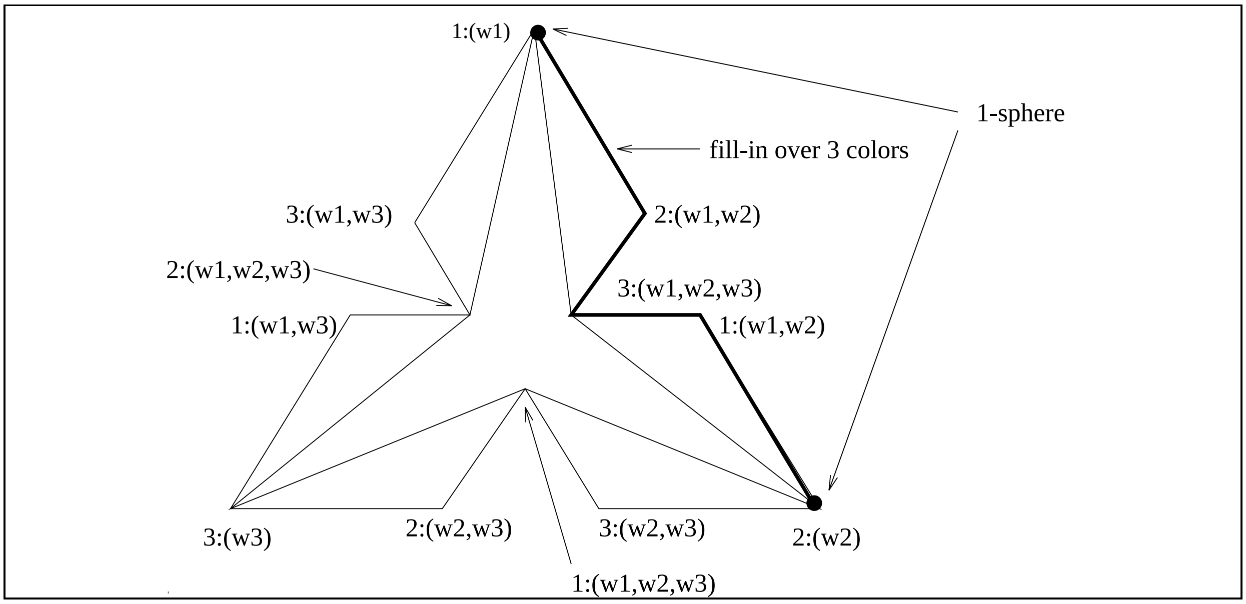

Figure 1: A possible view-graph for 3 processors using 2-consensus

Figure 1 is a possible view graph-for 3 processors using 2-consensus and read-write registers wait-free. The 1-sphere (2 vertices) consisting of $1 : (w1)$ (processor 1 read only its name) and $2 : (w2)$ resides on the 1-dimensional face across from $3 : (w3)$, but does not have a fill-in in the face. It does have a fill-in (path connecting the vertices) when considered inside the 2 dimensional face which is the whole complex.

Since the characterization is summarized by the growth function, models that have the same growth function are equivalent in the sense that the set of tasks solvable in each is the same. Indeed, for any combination of consensus object types, we can find an equivalent read-write active-resiliency model, but not vice-versa.

To derive the conditions, we rely on two novel techniques to reduce models with varying resiliency conditions and different consensus objects to the wait-free case.

For the sufficiency conditions, we have devised a new simplex convergence [3] protocol in the two resiliency models when processors may access set-consensus objects. The solution consists of convergence phases of wait-free read-write computation, with the exception that at the beginning of a phase the processors go through a narrowing step. To achieve the narrowing processors use the resiliency assumptions and the consensus objects. In this narrowing step, each processor suggests its input as a choice for the output, this results in processors obtaining one out of a limited number of distinct outputs chosen. Thus, protocol convergence for the models at hand is conceptually not much different from convergence in the read-write waitfree case, and therefore we may view it as a reduction to the waitfree model. When it is applied to the read-write waitfree model, our convergence protocol is quite different than the one in [3]. As a corollary, it provides a simpler algorithmic proof to the simplex agreement scheme that was proved in [3] by employing sophisticated algebraic-topology techniques.

For the necessary conditions, we use for the reduction the simulation-by-agreement technique introduced in [6]. In simulating a protocol written for one model by processors in another model, a relation between the protocol-properties of the simulated protocol complex and the protocol-properties of the simulator's protocol-properties is established. The view of a simulator include views of those processors in the simulated protocol which the simulator has observed to halt. Thus, since a simplex in the simulator's protocol complex corresponds to a unique execution of the simulated protocol, and therefore a simplex in the simulated protocol complex, we obtain a relation between the two complexes.

To make this paper self-contained we will re-explain the simulation here and expand on it. This expansion is crucial for the characterization. In a simulation-by-agreement, as the number of simulators decreases, each simulator has more freedom (i.e., nondeterministic choice) on how the simulation will proceed. Any deterministic rule that eliminates the nondeterminism can be considered a different simulation scheme, and so gives rise to different possible sets of simulated executions. The new realization in this paper is that by adding a preprocessing phase of simplex convergence to the simulation, we may produce executions (subject to some nontriviality conditions) of a given subdivided simplex in the simulated protocol's view-graph. This is the key that will allow as to refer to all spheres of a certain dimension, rather than just to special ones.

A corollary to our simplex convergence protocol with implications for programming distributed systems is the number of accesses to consensus objects depends only on the number of processors rather than the task. Another such result is that a program to solve a problem in the t-resilient model requires only one wait statement at the beginning of the codes of $n-t$ processors.

In formalizing the results, we have found that multiple-input tasks, rather than single-input tasks, and t-resiliency, rather than active-resiliency, introduce no conceptual problems but clutter and complicate the exposition. All the essential difficulties and ideas are manifested when considering only single-input tasks and active-resiliency. As a result, we postpone discussion of the characterizing conditions of traditional resiliency and multiple-inputs until Section 4, where by then understanding how they fit in is easy.

The paper is organized as follows: We begin by providing a section on the model and topological preliminaries. In that section, we also define the notion of g-solids, which is the protocol-property of models with growth function g and we associate different g functions with different model parameters. In Section 3, we present the necessary and sufficient conditions. In a subsection of sufficiency, we present the the new simplex convergence protocol. The subsection on the necessity, presents the expanded simulation-by-agreement and shows that it implies that any protocol complex in the model has the protocol-properties. Next, in Section 4, we extend the result to multiple input tasks and to traditional t-resiliency, and we discuss various ramifications of the characterizations and the techniques we used. Our conditions may be also stated in terms of the first barycentric subdivision of the protocol complex. Indeed, the conditions of [1] may be viewed as being given in terms of these barycentric subdivision. Also, some of our corollaries may be derived in a direct algorithmic way, without going through complexes. We then summarize our results and discuss open questions in the concluding section.

This paper includes few ideas rather than a central one which can be isolated. Because any break-up of the ideas would be artificial, for lack of space, most of the technical work appears in the appendix.

2 Preliminaries

2.1 Model

We will not try to define here a distributed system in its full generality since this paper deals only with a restricted set of models. We assume that the reader is familiar with asynchronous read-write shared memory, and we add here the possibility that processors can access set-consensus object types (we say types to emphasize that we do not limit the number of instances of the types used). An (m, j) set-consensus object can be accessed by at most m not necessarily predeclared processors ([9] shows that this is w.l.o.g.), each submitting some input. The object replies by returning to each processor one of at most j distinct inputs.

A model also includes a scheduler. A scheduler determines which of the processors that did not halt will be scheduled to execute its operation next. An *execution* is an infinite sequence of global states or a finite sequence in which all processors halt. Each two consecutive states in the sequence correspond to an allowed transition in the model.

Thus, our definition of execution does not include fairness.

We may impose fairness constraints on the scheduler. The type of constraints we will investigate in this paper are constraints on the number of processors that the scheduler may indefinitely ignore in favor of executing operations of other processors. A processor that is never scheduled is called *sleeping*. A processor that is scheduled at least once but does not halt is called *active*. In the traditional resiliency model, any processor that is indefinitely postponed from some point on is *faulty*. In the active-resiliency model a faulty processor is one that is indefinitely ignored after being scheduled at least once. A system in which each processor is never ignored indefinitely is a system with no faults; such systems can solve any centrally computable task. A system in which any number of processors may be indefinitely ignored in favor of others is a wait-free system. A system in which at most t processors, whether sleeping or not, may be ignored indefinitely is a t-resilient model. Finally, a model in which of the i active processors at most $r(i)$ can be ignored indefinitely, while any sleeping processor may be ignored indefinitely, is our new (active) $r(i)$-resilient model. To stay on intuitively sound ground, we restrict our investigation to functions $r(i) \leq i$ which are monotonically nondecreasing and which increase by at most one at a time.

2.2 Tasks and Solvability

A *task* is a point to set mapping that takes any set S of processors with associated input values and returns sets of outputs. Each output value in an output set is associated with a distinct processor in S.

A task is solvable in a given model if there is a protocol in the model such that in all executions in which the scheduler does not violate its constraints, all processors halt (and consequently the execution is necessarily finite). Moreover, in all prefixes of executions in which a set of processors woke up and halted before any processor not in the set woke up, the set of outputs matches the set of inputs through the task mapping.

Of particular interest in this paper will be the $(i, g(i))$ *narrowing* task $i \in N$, in which if i processors halted before the rest woke up, then each processor outputs one of at most $g(i)$ distinct input values.

2.3 Topological Preliminaries

The final (halting) state of a processor in a protocol is modeled as a vertex, also referred to as a *view*. A set of vertices is a *simplex* if the associated views arise from the same execution of the protocol. A subset of vertices of a simplex is called a *face* of the simplex. Obviously a face of a simplex is a simplex. A set of simplexes C is a *complex* if for any simplex $s \in C$ every face $s' \subset s$ is also in C. A *principal* simplex of a complex is one that is contained in no other simplex. The dimension of a simplex is the the number of its vertices minus one. In this paper, we deal only with *pure* complexes, in which all principal simplexes have the same dimension. The dimension of a pure complex is the dimension of a principal simplex. The dimension of a complex is denoted by a superscript, for example C^d denotes a d dimensional complex. The same notation holds for simplexes.

A pure n dimensional complex is *chromatic* if it is $n+1$ colored, and every principal simplex is colored by $n+1$ distinct colors. The color of a vertex, denoted $id(v)$, represents the ID of the processor whose view is identified with the vertex. The *carrier* of a point p in a complex C^n, denoted $carrier(p, C^n)$, is the smallest simplex of C^n that contains p. For an embedded simplex $s = (v_0, ..., v_n)$, the *barycenter* of s is the point $\hat{s} = (v_0 + \ldots + v_n)/(n+1)$. The first barycentric subdivision of C^n, denoted $bsd(C^n)$, is a complex whose vertices are the barycenters of simplexes in C^n, and a simplex of the barycentric subdivision is any set of barycenters of simplexes related by containment, all of which are subsimplexes of the same simplex in C^n.

A *simplicial map* $\mu : C \to C'$ is a function from nodes of C to nodes of C' which preserves adjacency. Namely, for a simplex of vertices $v_1, ..., v_i$ in C, $\mu(v_1), ..., \mu(v_i)$ is a simplex in C'. A simplicial map from one chromatic complex to another is chromatic if it preserves colors, namely, if $id(v) = id(\mu(v))$. A point to set mapping from one chromatic complex A to another chromatic complex B is chromatic if the output as the same colors as the input and any simplex in the image of $s^i \in A$ is of dimension i.

All simplexes and complexes have *embeddings* in Euclidean space of high enough dimension so that the vertices of each simplex are affinely independent and so that the intersection of any two simplexes is either a face or empty. We denote the embedded version of a complex C as $|C|$. Complex $|B|$ is a *subdivision* of $|A|$ if for each simplex $b \in B$, there exists a simplex $a \in A$ such that $|b| \subseteq |a|$, and $\cup_{b \in B}|b| = |A|$. We denote a subdivision of complex A as $subd(A)$. From here on we assume all simplexes and complexes are embedded; thus, we will always leave off the bars.

Given a simplex s of complex C, the *star* of s in C, denoted $st(s, C)$, is the union of all simplexes in C that contain s. The *open star*, denoted $\overset{\circ}{st}(s, C)$, is $st(s, C)$ minus any simplexes of C that do not intersect s. The *linking sphere*, denoted $ls(s, C)$, is the corresponding star minus the open star. When the context is obvious, we will denote the star as $st(s)$, and likewise for the open star and linking sphere. Given a set S of vertices of a complex C, the subcomplex of C *induced* by S is the complex that consists of all the simplexes in C, all of whose vertices are in S.

2.4 Nonstandard Definitions

2.4.1 Topological Description of Tasks

Given a task on $n+1$, processors we model the input and output as n dimensional complexes I^n and O^n, respectively [2]. The relation between input and output sets is now a relation between complexes and is denoted by Δ. Thus, topologically, a task is a 3-tuple $\langle I^n, O^n, \Delta \rangle$ where Δ is a relation between two chromatic complexes I^n and O^n of inputs and outputs, respectively, which maps from a simplex of I^n to a set of simplexes of O^n.

2.4.2 Spheres and Generalized Solids

A *boundary* of a subdivided simplex S^n, called a *sphere*, is the subcomplex induced by all the vertices that are on faces of S^n of dimension less than n. Given a simplicial image $\mu(S)$ of a sphere S in complex A, we will say that $\mu(S)$ has a fill-in in A if there exists a subdivided simplex B with boundary S and a simplicial map $\bar{\mu} : B \rightarrow A$ such that $\bar{\mu}$ agrees with μ on S [2].

We want to generalize a subdivided simplex into a "subdivided simplex with missing simplexes." At the same time, we would like to retain the notion of a face. We do this by taking a point to set map from a simplex to the complex. The image of a face of the simplex will informally be our new notion of a face. Finally, we capture by a function g the fact that a sphere of a certain dimension may only have a fill-in within a higher dimensional complex.

The following definition plays a crucial role in this paper.

Given a chromatic n-dimensional complex C^n and a function $g : N \rightarrow N$, we say that the complex C^n is a *g-solid* of dimension n if:

1. There exists a chromatic point to set map δ from a chromatic n-simplex s^n to C^n such that $C^n = \delta(s^n)$ and if $s_1 \subset s_2 \in s^n$, then $\delta(s_1) \subset \delta(s_2)$.

2. Any simplicial image G of a sphere S in C^n, with S of dimension $\leq g(n+1)$ has a fill-in in C^n.

3. The linking sphere $ls(s^q, C^n)$ of any q-dimensional simplex $s^q \in C^n$ is inductively a g-solid of dimension $n-(q+1)$.

4. The complex $\delta(s^q)$ for any $s^q \in s^n$ is inductively a g-solid of dimension q.

Given a δ map for the first condition, we will also say that the complex C^n is a g-solid of dimension n *with underlying face structure* δ.

To see that this definition generalizes a subdivided simplex, we have the following theorem.

Theorem 2.1 *An n-dimensional subdivided simplex is an f-solid of dimension n where $f(i) = i-1$.*

2.4.3 Narrowing Powers of Set-Consensus and Resiliency

In this subsection we will derive the narrowing function of a model with $r(i)$-resiliency with access to a set of set-consensus types $\{O_1 = (m_1, k_1), ..., O_l = (m_l, k_l)\}$.

Proposition 2.1 *A wait-free model over n processors with access to (m, j) set-consensus can solve the task $(i, (i \text{ div } m)j + \min\{j, i \text{ mod } m\})$.*

Proposition 2.2 *In a wait-free system with i processors with access to a set of set-consensus types $\{O_1 = (m_1, k_1), ..., O_l = (m_l, k_l)\}$, let n_i be the minimum integer such that (i, n_i) is solvable. Let m_j be such that $m_j \leq i$ and $\forall h \; m_j/k_j \geq m_h/k_h$, then for $i_1 = i \bmod m_j$, $n_i = (i \text{ div } m_j)k_j + n_{i_1}$.*

Proposition 2.3 *A system with n processors with $r(i)$ resiliency and with access to a set of set-consensus types $\{O_1 = (m_1, k_1), ..., O_l = (m_l, k_l)\}$ may solve for each i $(i, n_{r(i)+1})$ (where $n_{r(i)+1}$ is defined in Proposition 2.2).*

The difficulty in proving Proposition 2.3 is that the narrowing function is dynamic with respect to the number of processors in the system. To accommodate this constraint, we propose the strong $r(i)+1$ election problem, which is essentially a one-shot l-exclusion problem [10], where l increases as the number of non-sleeping processors grows. The iterated problem is proposed so as to capture the case that the number participants may decrease. We relegate the proof to the appendix.

3 Necessary and Sufficient Conditions for Solvability

Theorem 3.1 *Let M be a model that has a certain resiliency function and access to set consensus objects such that $(i, g(i)+1)$ is solvable. Let $T = (I^n, O^n, \Delta)$ be a task over $n+1$ processors. Then T is solvable in M iff there exist a g-solid complex Φ^n of dimension n with an underlying face structure δ, and a simplicial chromatic map $\mu : \Phi^n \rightarrow O^n$ such that $\mu(\delta(s^q)) \subseteq \Delta(s^q)$, $\forall\ s^q \subseteq s^n$.*

The theorem is based on the existence of Φ^n. The complex Φ^n is a view-graph of a protocol that solves the task (if one exists). In the necessity subsection below, we prove that any protocol in a model that can solve $(i, g(i)+1)$ (by employing resiliency and set-consensus objects) must have a view-graph complex that is a g-solid. The sufficiency conditions will let processors converge to a simplex over Φ^n and then decide on outputs through the map μ.

Relying on the existence of Φ^n is unavoidable. There is no algorithm to construct it. In [11] it was proved that the question of task solvability is undecidable even for single-input tasks for three processors.

3.1 Sufficiency: Generalized Simplex Agreement

The problem of simplex agreement over subdivided simplex was first proposed and solved by Herlihy and Shavit [3]. In simplex agreement, processors are given a chromatic complex and must each choose a node of their own color from a common simplex. The nontriviality condition is that a set S of processors that halted before any other processor started has a choice of a simplex on the face defined by the colors in S.

We first show that when Φ^n is $(i-1)$-solid there exists a desired subdivided simplex in Φ^n. For all the vertices v_k^0 of I^nm we choose a distinct vertex of $\delta(v_k^0)$. For each pair of vertices, one in $\delta(s_j^0)$ and one in $\delta(s_i^0)$ (a 1-sphere), there must exist an fill-in (a path) that is inside $\delta((s_j^0, s_i^0))$. We choose such a fill-in. For each triple, taking the fill-in of each of the pairs yields a 2-sphere that has a fill-in inside the δ of the simplex of the triple; and so on. Thus, we obtain a subdivided simplex $B^n \subseteq \Phi^n$. We conduct the convergence on this subdivided simplex B^n and map through μ to O^n.

When we have a general g-solid rather than a $g = i-1$-solid, this procedure may break down even at the first step since there is no guarantee that there is a path between $\delta(s_j^0)$, and $\delta(s_i^0)$ inside $\delta((s_j^0, s_i^0))$. (In fact, if the model is not wait-free for two active processors, or if we have $(2,1)$ set-consensus objects, such a path will not exist.)

The sufficiency condition is defined in the following theorem.

Theorem 3.2 *Let $GSA = (I^n, \Phi^n, \delta)$ be an $n+1$ processors single-input finite output task. Then GSA is solvable in a model of certain resiliency with set-consensus objects in which $i, i = 1, ..., n+1$ processors can solve $(i, g(i)+1)$ if Φ^n is g-solid with underlying face structure δ.*

We call the GSA task Generalized Simplex Agreement. The solution to the generalized simplex agreement problem, and the proof of Theorem 3.2 appear in the appendix.

3.2 Necessity

Theorem 3.3 *Let (I^n, O^n, Δ) be an $n+1$ processor single-input task solvable in a certain resiliency function model with access to set consensus in which $i, i = 1, ..., n+1$ processors can solve $(i, g(i)+1)r$. Then there exist a g-solid complex Φ^n with underlying face structure δ, and a chromatic simplicial map $\mu : \Phi^n \to O^n$ such that $\mu(\delta(s^q)) \subseteq \Delta(s^q), \forall s^q \subseteq s^n$.*

The crux of the proof involves a (deterministic) wait-free read-write simulation of a protocol for (I^n, O^n, Δ). Since the simulation is wait-free read-write, the view-graph of the simulators is a subdivided simplex [6, 12, 7]. When a simulator terminates, its view is guaranteed to contain at least one final code view, which induces a mapping from simulator views to sets of code views. This is a mapping to the first barycentric subdivision of the code view complex. Since an execution of the simulation produces a run of the simulated protocol, this mapping is simplicial. By mapping each barycenter to a vertex of its carrier in the simulated protocol view complex, we induce a simplicial map from the simulation view-graph (a subdivided simplex) to the task protocol view-graph, thus providing a filled-in image of a sphere in the protocol view-graph.

Inductively, when given a particular i dimensional image of a sphere, we will show that we have a simulation scheme whereby i simulators produce the views of the image of each of the faces defining the sphere. Thus, with all combinations of i processors, we produce the image of a sphere. When $i+1$ simulators are encountered, they necessarily produce a fill-in of the image of the sphere.

The number of processors that may simulate a protocol and still guarantee that any one simulator will have a final code view depends on the set-consensus objects in the model and the resiliency of the model. For our purposes, this dependence is manifested entirely through the narrowing function g, and that is what determines the g-solidity of the code view-graph. We may generate a $g(i)$ dimensional sphere with only $k = min_j\{g(j) = g(i)\} + 1$ codes. Each image of a face of the sphere can be simulated by $g(k)$, but not $g(k)+1$, simulators with the guarantee that a halting simulator will see a code view. For $g(k)+1$ simulators, if we have k codes we are guaranteed that a halting simulator has code view in its view. Thus we prove that a $g(k)$ image of a sphere has a fill-in in dimension k. Since the protocol simulated is assumed to be correct, it provides the map μ from Φ^n to O^n that satisfies the theorem.

The full proof and details of the simulation technique appear in the appendix.

4 Extensions and Ramifications

Omitted in this abstract.

5 Conclusions

We have introduced the new model of active-resiliency. We have shown how to associate, with either the traditional-resiliency model or the active resiliency model in which set consensus objects are used, a single narrowing function g. We have shown that this function g completely characterizes the power of the model. Moreover, we have stated the protocol-properties of complexes in the model as a conceptually simple natural extension of the protocol-properties of the read-write waitfree model [2, 3]. This has resolved the next major open question in the area following the results of [2, 3]. This work is another step towards a fundamental understanding of asynchronous distributed computing.

Technically, we have contributed some important tools the area's arsenal. We have presented a new simplex convergence protocol. We have expanded the simulation-by-agreement of [6], and we have posed and solved the generalized narrowing problem. We have shown that the characterization and the tools have interesting ramifications (omitted in this Abstract). Moreover, any previous results concerning such models are a simple corollary to this paper (e.g. [13, 8] and parts of [14]).

What next? There is no hope of a simple characterization of models using arbitrary objects. With the most general definition of an object, objects are as varied as topological structures. Nor is it obvious that general objects are interesting. Specialization to specific families is called for. In the same vein, since the solvability of tasks is undecidable [11], identifying natural families of decidable tasks may be interesting. Also, the question of long-lived objects has not been touched. Finally, tasks may not capture all aspects of synchronization. As a case in point, the reader may have noticed that we did not pose the strong leader election problem (essentially capturing the one-shot exclusion problem) as a task.

References

[1] O. Biran, S. Moran, and S. Zaks, "A Combinatorial Characterization of the Distributed Tasks which Are Solvable in the Presence of One Faulty Processor," in *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing*, pp. 263–275, 1988.

[2] M. Herlihy and N. Shavit, "The Asynchronous Computability Theorem for t-Resilient Tasks," in *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pp. 111–120, 1993.

[3] M. Herlihy and N. Shavit, "A Simple Constructive Computability Theorem for Wait-free Computation," in *Proceedings of the 26th ACM Symposium on the Theory of Computing*, 1994.

[4] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[5] S. Chaudhuri, "More *choices* allow more *faults*: Set consensus problems in totally asynchronous systems," *Information and Computation*, vol. 105, pp. 132 – 158, jul 1993. supercedes 1990 PODC version.

[6] E. Borowsky and E. Gafni, "Generalized FLP Impossibility Result for t-Resilient Asynchronous Computations," in *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pp. 91–100, 1993.

[7] M. Saks and F. Zaharoglou, " Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge," in *Proceedings of the 26th ACM Symposium on the Theory of Computing*, pp. 101–110, 1993.

[8] M. Herlihy and S. Rajsbaum, "Set Consensus Using Arbitrary Objects," in *Proceedings of the 13th ACM Symposium on Principles of Distributed Computing*, pp. 324–333, 1994.

[9] E. Borowsky, E. Gafni, and Y. Afek, "Consensus Power Makes (Some) Sense!," in *Proceedings of the 13th ACM Symposium on Principles of Distributed Computing*, pp. 363–372, 1994.

[10] Y. Afek, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, "First-in-First-Enabled l-Exclusion," in *Proceedings of the 4th International Workshop on Distributed Algorithms*, 1990.

[11] E. Gafni and E. Koutsoupias, "3-processor tasks are undecidable." To Appear in 14th Symposium on Principles of Distributed Computing, 1995.

[12] E. Borowsky and E. Gafni, "Immediate Atomic Snapshots and Fast Renaming," in *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing*, pp. 41–51, 1993.

[13] E. Borowsky and E. Gafni, "The Implication of the Borowsky-Gafni Simulation on the Set-Consensus Hierarchy," Tech. Rep. 930021, University of California, Los Angeles, 1993.

[14] T. Chandra, V. Hadzilacos, P. Jayanti, and S. Toueg, "Wait-Freedom vs. t-resiliency and the robustness of wait-free hierarchies," in *Proceedings of the 13th ACM Symposium on Principles of Distributed Computing*, pp. 334–343, 1994.

[15] H. Attiya, N. Lynch, and N. Shavit, "Are Wait-Free Algorithms Fast?," in *Proceedings of the 31st Symposium on Foundations of Computer Science*, 1990.

[16] D. Dolev, N. Lynch, S. Pinter, E. Stark, and W. Weihl, "Reaching Approximate Agreement in the Presence of Faults," *Journal of the ACM*, vol. 33, no. 3, pp. 499–516, 1986.

[17] A. Fekete, "Asymptotically Optimal Algorithms for Approximate Agreement," in *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing*, 1986.

[18] G. Frederickson and N. Lynch, "Electing a Leader in a Synchronous Ring," *Journal of the ACM*, vol. 34, no. 1, pp. 98–115, 1987.

A Appendix

A.1 Strong $r(i)+1$ Election Problem

The difficulty in proving Proposition 2.3 is that the narrowing function is dynamic with respect to the number of processors in the system. Intuitively, what we would like is, starting with a

$g(i)$-exclusion dynamic protocol, to let processors in the critical section access the objects and post outputs for processors outside the critical section.

To capture the one shot dynamic exclusion protocol we pose the following problem:

The Strong $r(i)+1$ Election Problem: This problem is the $(i, r(i)+1)$ narrowing problem with the additional requirement that each processor reveals its input in a last write operation before it halts.

Later in the exposition we will have a sequence of such critical sections and we want the narrowing at each stage in the sequence to be a function only of the number of processors that will ever reach the stage. Thus we propose the iterated version of the problem.

The Iterated Strong $r(i)+1$ Election Problem: Depending on their local state after solving the strong $r(i)+1$ election problem (where decision now may be committed by writing the output to shared memory), processors may either proceed to another stage of performing strong election or halt. In an execution where set S halted altogether without the others waking up, if j processors proceeded to the second stage they narrowed to at most $r(j)+1$ inputs that were revealed in the last write of the second stage.

Obviously, a solution to the iterated strong election problem is a solution to the non-iterated problem. Thus we will present a solution to the iterated problem that views the first election phase as an uninterpreted computation in which a processor either halts or proceeds to the election stage.

Here we chose to use the participating set algorithm of [9].

Solution to the Iterated Strong $r(i)+1$ Election Problem, in a read-write n processor model with $r(i)$ resiliency: A token is a place holder for a value. Processors descend through a sequence of n levels. Processor j starts by signalling that it is awake (i.e., the first operation is write). It then proceeds through the uninterpreted computation (the previous phase(s) of the iterated strong election). If processor j is to halt and not proceed to the election (the current strong election phase), it signals its halt, and halts (i.e., the last operation is a write). Otherwise, it enters the election phase by placing a token holding ID j at level n. Inductively, assume processor j has placed its token at level k. It then takes a snapshot of all levels. If the number of tokens it sees in level k or below is less then k, it drops and places the token at level $k-1$. Otherwise, it writes "j-stopped" on the token.

Processor j orders all processors. Processors with tokens that have not stopped or processors that woke up but did not place a token are considered at level 0. The processors are ordered lexigraphically by (level, token-ID). Processor j then waits for one of the following conditions: either processor j's location in the order is $\leq r(i)+1$, where i is the number of processors that processor j observed to wake up and not halt without proceeding to the election, or another processor wrote its input to the election. In the latter case processor j adopts that output and halts. In the former case (to be called "entered the critical section"), it adopts its own input as output, reveals its input, and halts.

We use the iterated strong election problem as a subroutine to the simplex agreement algorithm.

Proof of Proposition 2.3

To prove Proposition 2.3, we use the solution to the election problem. Processors submit their

ID for the strong election. A processor that enters the critical section does not reveal its input but proceeds to a narrowing phase through the consensus objects. It adopts its output from the narrowing phase, reveals it, and halts.

By the election algorithm, at most $r(i)+1$ processors will reach the consensus phase and therefore the number of outputs is at most $n_{r(i)+1}$. Since the election algorithm terminates and the consensus object phase is wait-free, at least one processor will reveal its output and therefore all processors will halt. $\square$

A.2 Sufficiency

The simplex agreement algorithm proceeds in phases. At the beginning of a phase, processors narrow the number of starting points using their ability to solve $(i, g(i)+1)$ set-consensus. Then the processors perform ϵ-agreement [15, 16, 17] on the appropriately determined fill-in of a sphere. The narrowing is needed to reduce the number of starting points far enough to ensure such a fill-in exists. The sphere is determined inductively. Each two starting points (a 1-sphere) uniquely determine a fill-in. Each three starting points, together with the fill-ins of their pairs, determine an image of a 2-sphere. In turn, the images of the 2-spheres uniquely determine a fill-in, and so forth. Thus each image of a face of a subdivided simplex is uniquely determined by the images of the corners of the face. These corners are our starting points. This means that no processor need know the sphere in advance. As each processor joins the ϵ-agreement algorithm, the fill-in is enlarged consistent with already determined faces. After the ϵ-agreement phase, at least one processor has decided, and the rest of the processors go on to the next phase in the linking sphere of the processors that have decided. No processor knows the linking sphere, but each group defines a fill-in that contains the fill-in defined by any subgroup. Precisely as in the first phase, the part of C^n on which processors converge grows consistently with prior determinations.

We now supply the details.

The first problem we face is that, unlike the read-write wait-free case where $g(i) = i - 1$, our fill-in now lives in a complex of dimension i that may be bigger than the number of starting points $g(i)+1$. How do the processors determine the fill-in on which to converge? A useful example is a wait-free system employing $(3, 2)$ in which four processors participate in convergence. The processors may narrow to three starting points, so they know at least three processor colors, but since they need to converge on a fill-in that lives in three dimensional complex, how do they agree on another color? In the meantime, another processor may join up, but how do they agree on this fourth processor?

We show that we can do more than just narrow the starting points. We can have each starting point associated with a snapshot of the participating processors such that the three snapshots uniquely determine a set of at least four of the colors of the participating processors.

The generalized narrowing Problem: If a set S of processors wake up and terminate before the rest even start, then each processor i decides on $S_i \subseteq S$ such that:

1. $i \in S_i$.

2. $\forall p, q \; S_p \subseteq S_q$ or $S_q \subseteq S_p$.

3. There are at most $l \leq g(i) + 1$ distinct output sets.

4. The union of any $k \leq l$ distinct sets has cardinality $\geq g(k) + 1$.

Conditions 1 –3 simply specify the snapshot task followed by regular narrowing. The interesting requirement is the last, which limits the number of processors that can take small snapshots.

Lemma A.1 *The generalized narrowing problem is solvable in a system in which n processors can solve $(i, g(i) + 1)$ by the availability of set-consensus objects and resiliency.*

Proof We build n stages i, $i = 1, ..., n$. Each stage i is accessed by at most i processors and the processors accessing the stage perform a $g(participating) + 1$ narrowing. A processor starts by writing its name into shared memory. It then takes a snapshot (this will satisfy the first two conditions). If its snapshot size is i, it starts to execute stage i. Inductively, upon finishing stage k, it takes a new snapshot $snap^k$ to update its view of the processors that woke up. It then proceeds to stage $k + 1$, taking as input to the stage the output (which is a set of processors) it obtained at stage k or in $snap^k$, whichever includes more processors. The output obtained in stage n is adopted as the final output.

If the choices available at stage i satisfy the generalized narrowing conditions for i processors, then the elimination of any choice still leaves a set of choices which satisfies the generalized narrowing conditions for i processors. At stage $i + 1$, outputs of stage i may be eliminated and new outputs chosen. This poses no problem since any new output to stage $i + 1$ is of size greater than i.

It is not necessarily the case that stage i does $(i, g(i) + 1)$ narrowing, since the number of active processors may be greater than i so some processors will not participate in stage i. The idea is that inductively the snapshots of all processors (say l of them) that finished stage i before a snapshot of greater size exists will satisfy the generalized narrowing conditions for l processors. All processors that finish stage i after a larger snapshot exists will discard their choice and adopt a snapshot larger than size i as input to stage $i + 1$.

$\square$

This solves the narrowing phase of the generalized simplex agreement. We now turn to the ϵ-agreement protocol.

ϵ-Agreement: The problem of ϵ-agreement is for n processors waking up at arbitrary points in Euclidean space to halt with points that are within ϵ of each other, such that each point is in the simplex defined by the input points of processors that woke up.

The ϵ-agreement problem was formulated in [16] and solved in [15]. Here we supply an inefficient but rather simple-minded solution. This will divorce our exposition from any idiosyncrasies in the literature on this problem. Imagine a processor P_i waking up at point p_i. P_i alternately adds a weight $w(p_i)$ at p_i and takes a snapshot of the whole system to see the total weight in the space. We take the function w to be one in which $w(p_i)$ decreases as the distance between p_i and the origin increases and if w decreases fast enough with respect to the distance so that there is a total weight of the system $W(\epsilon)$ such that any addition of n weights changes the center of gravity by less then $1/2\epsilon$. Processors wake up and add weights until they observe a total weight of $W(\epsilon)$. They then choose as output the center of gravity point that they observed. By the specification, these centers will be within ϵ of one another.

After the processors conduct ϵ agreement, few of the processors will halt and the remaining will start a new phase of the algorithm, namely, converging on the linking sphere of the simplex determined by the halted processors. After presenting a lemma providing a topological basis for our ϵ, we present Lemma A.3, the heart of the simplex agreement procedure.

Lemma A.2 *Let $\mathcal{A}$ be an open cover of a compact space. Let δ be its Lebesgue number. If $x \in A \in \mathcal{A}$, and $\forall B \neq A$ $x \notin B$, and $y \notin A$, then the minimum distance between $1/4\delta$ balls around x and y is at least $1/2\delta$.*

Lemma A.3 *Let $|\mathcal{S}^n|$ be an embedding of a finite subdivision of S^n in R^{n+1}. Then there exists a function $s : |\mathcal{S}^n| \to \mathcal{S}^n$ and an $\epsilon > 0$ that maps $x \in |\mathcal{S}^n|$ to a simplex $s(x) = s^m \in carrier(x, |\mathcal{S}^n|)$ such that for any subset $N(\epsilon)$ with diameter $\leq \epsilon$, $\bigcap_{y \in N(\epsilon)} s(y) \neq \emptyset$ and $\bigcup_{y \in N(\epsilon)} s(y) = s^l$ for some $s^l \in \mathcal{S}^n$.*

Proof We define the function $s(x)$ inductively in the dimension of $s(x)$. To define the part of the space $\{S_0 | x \in S_0 \text{iff} dim(s(x)) = 0\}$, we take the open cover of $|\mathcal{S}^n|$ by the open stars of the zero-dimensional simplexes of S^n. We take the Lebesgue number of the cover $\delta_0 > 0$ (δ_0 is such that for each subset N_{δ_0} of diameter $\leq \delta_0$, there exists an open star A such that $N_{\delta_0} \subseteq A$), and define $S_0 = \{x | \exists {s^0}_x \; dist(x, {s^0}_x) \leq 1/4\delta_0\}$. It is easy to see that each $x \in S_0$ is in S_0 as a result of a unique ${s^0}_x$. For $x \in S_0$, we define $s(x) = {s^0}_x$.

Inductively we have defined $S_0, ..., S_{i-1}$. To define S_i we take the open cover of $|\mathcal{S}^n|$ that consists of the open stars of the i-dimensional simplexes together with $interior(S_k)$, $k = 0, ..., i-1$. Let its Lebesgue number be δ_i. Then $S_i = \{x | \exists {s^i}_x \; dist(x, {s^i}_x) \leq 1/4\delta_i, \; x \notin \cup_{j=0}^{j=i-1} S_j\}$. It is easy to see that each $x \in S_i$ is in S_i as a result of a unique ${s^i}_x$. For $x \in S_i$, we define $s(x) = {s^i}_x$.

We now let $\epsilon = 1/4 \min_{i=0}^{n}(\delta_i)$

The union and intersection properties will be implied if we prove that, for any x and y, if $dist(x, y) \leq \epsilon$, then $s(x)$ and $s(y)$ are related by containment. This will follow easily from the construction of the function $s(x)$ and from Lemma A.2.

If we now have x and y such that $dist(x, y) < \epsilon$, $s(x) \not\subseteq s(y)$, $s(y) \not\subseteq s(x)$, then w.l.o.g. let dimension $s(x) = k$ be less than or equal to dimension of $s(y)$. Then in the cover considered in step k, all points in $s(x) - \cup_{i=0}^{i=k-1} S_i$ belong solely to the open star of $s(x)$, and points in $s(y)$ are not in the open star of $s(x)$, contradicting the fact that $\epsilon < 1/2\delta_k$.

□

Figure 2 shows the partitioning of a two-dimensional complex into regions. Though other mappings may yield larger values of ϵ, this scheme has been helpful for understanding the convergence.

Proof of Theorem 3.2

The algorithm that solves the generalized simplex agreement problem starts by having the processors solve the narrowing problem. Each processor then posts the snapshot it obtained from the narrowing problem and starts convergence from a vertex in δ(*lowest ID processor in its snapshot)*. Inductively, when the number of snapshots posted is $g(i)$, each distinct $g(i) - 1$

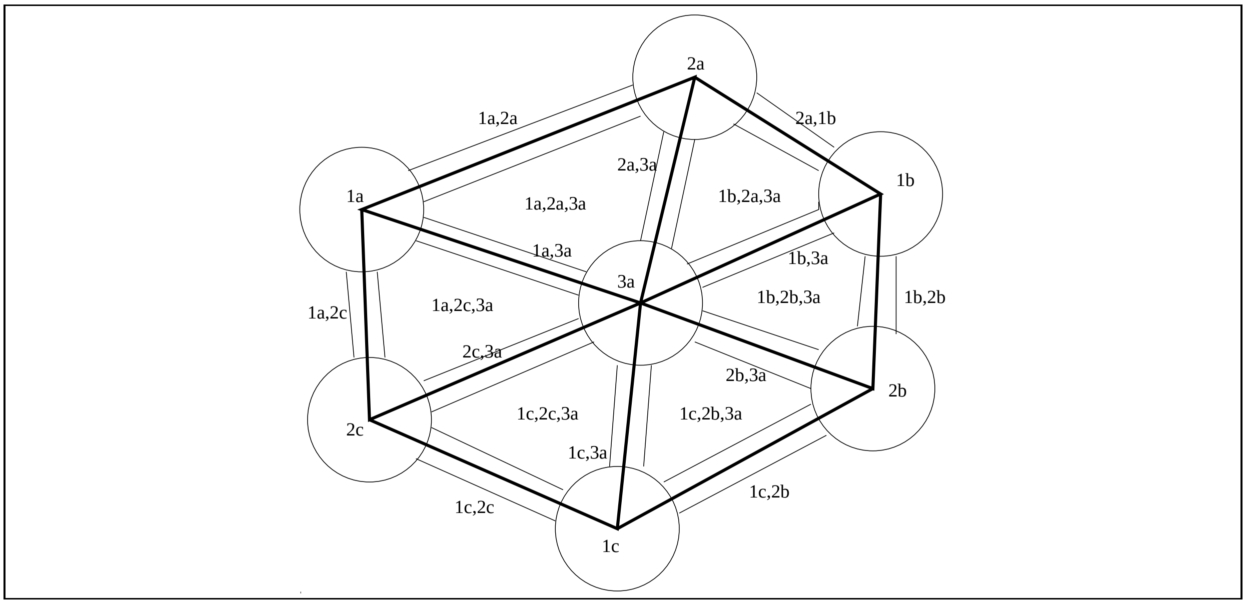

Figure 2: Example of complex partitioning and map f.

of them determines a fill-in. Inductively, when taken together, all the fill-ins for the $g(i)$ groups of $g(i)-1$ vertices determine an image of a sphere. The fill-in of the sphere in which the $g(i)$ processors will converge is pre-agreed to be the sphere that lives in the subcomplex which is the δ of the simplex associated with the union of all the colors in the $g(i)$ snapshots. Our generalized narrowing guarantees that the size of the union is at least i.

A processor i converges to a point x in the appropriate fill-in. (Notice that a fill in implies the existence of a subdivided simplex. We converge over the subdivided simplex and than map back into the fill-in.) It then posts the simplex $s(x)$ in shared memory. (The function s is specified by lemma A.3). It takes a snapshot of all the other simplexes posted and takes the intersection. If its color is in the intersection, the processor decides the vertex, posts the fact that it is halting, and halts. By Lemma A.3, since the intersection is nonempty, at least one participating processor will halt and thus will not continue to the next phase. If the processor does not halt, it takes the union of all the posted simplexes. Again by the lemma, this union is a simplex us_i. The processor will then consider all the vertices of us_i except that of its own color (if such exists), possibly obtaining a new simplex usl_i. This is the simplex of vertices that may have been decided. This concludes the first phase.

The next phase again starts with processor i posting the tuple (i, usl_i) in shared memory. Processor i then takes a snapshot of the tuples and proceeds to a generalized narrowing step. The iterated strong election algorithm now comes into play. It is easy to see that iterated election along with generalized narrowing results in an algorithm that does generalized narrowing just with respect to the processors that proceeded to the second phase.

Obtaining a snapshot $\{(i_1, usl_{i_1}), ..., (i_l, usl_{i_l})\}$, the processor identifies a pre-agreed vertex of the color of the lowest ID in this snapshot which is in the linking sphere of $\cap_{j=i_1}^{j=i_l} usl_j$. For two points like this which result from different snapshots, we find a fill-in in the linking sphere of the intersection of all the *usl*'s in the two snapshots. We take the fill-in inside the smallest $\delta(s^q)$ (with respect to C^n, i.e., "face") that contains all the vertices in the *usl*'s and all the colors of IDs in the snapshots. The generalized narrowing phase guarantees that such a fill-in exists.

□

A.3 Necessity: Read-Write Simulation-by-Agreement

In [6, 13], we introduced a novel scheme called *simulation by agreement.* We have shown how to simulate in the wait-free read-write model, codes designed for a model employing resiliency and consensus. We repeat and expand on it here.

A.3.1 Wait-free Read-Write Simulation-by-Agreement

We are given n codes (a protocol) each with respective input, written for a model with certain resiliency and with access to certain consensus object types such that it can achieve $(i, g(i)+1)$ narrowing. We assume w.l.o.g. that the protocol is full-information [18]. Each read is a snapshot. Each write is the entire history. No writes are ever overwritten, and each access to consensus object is immediately followed by a write.

The codes of a protocol include commands of the type *write, read, submit(consensus-object-id, value)*, and *receive-reply(consensus-object-id, value).* The last two commands in a code always appear as a pair with matching *consensus-object-id*, with the former immediately preceding the latter. The last command of a code is a *halt.*

The protocol codes with give inputs are placed in the shared memory of the simulation system ($g(i)+1$ simulators working wait-free with read-write shared memory). The remainder of the simulation system shared memory is partitioned into "sections" according to the commands in the codes. Each *read* command, *write* command, and *receive-reply* command corresponds to a section. In addition, for each *object-id*, there is a single section for the set of all *submit* commands for that *object-id.*

Each section provides a place to conduct an *agreement-protocol* by the simulators. An instance of an agreement protocol is the participating set algorithm of [12].

Participating set agreement protocol [12]:
A token is a place holder for a value. With $g(i)+1$ simulators active, each simulator goes through at most $g(i)+1$ levels. Simulator j starts by placing its token at level $g(i)+1$. Inductively, assume processor j has placed its token at level k. The processor then takes a snapshot of the levels. If the number of tokens on levels k and below is less then k, the processor moves its token to level $k-1$. Otherwise, it writes "j-stopped." At this point processor j time shares between executing agreement protocols for other codes, and taking a snapshot of the section. An agreement terminates by the following rules.

1. For all sections but a *submit* section a processor waits until there exists a level on which a processor has stopped such that all processors with tokens on or below that level have announced they have stopped. The processor then picks the value on the face of the token

belonging to the lowest ID processor on the lowest nonempty level to be its agreement value.

2. For a *submit* section that corresponds to (m, j) set consensus, the processor waits until there exists a level on which a processor has stopped such that the number of processors on or below that level that have not stopped is less than j. Again the processor picks as its agreement value the value on the face of the token belonging to the lowest ID processor that is at the lowest nonempty level (whether this processor has stopped or not).

We have the following simple lemma.

Lemma A.4 *The number of distinct agreement that result from a* submit*-section corresponding to (m, j) object, is at most j.*

A simulator takes a snapshot of the whole memory. It interprets every agreement section in which it can see an agreement value as the command executed with the agreement value as the outcome. Inductively, a simulator will have a prefix of histories for all codes. The simulator associates with code histories a unique state of the shared memory and a unique local state for each code (and consequently the position of the code program counters). The simulator chooses the agreement corresponding to a program counter command in which there is no agreement value and executes it. The input it brings to the agreement is as follows.

1. If the command is *write* it interprets the local state of the processor code and submits this as the write value. (This step was introduced for the sake of modularity. All simulators will bring the same input value to a *write* agreement, and consequently the agreement protocol is superfluous).

2. If the command is a *read*, the simulator submits as an input the state of the shared memory it interprets. Different simulators may bring in different input values to a common *read* agreement. Inductively, all of these input values are legal possible extensions to the history the code read.

3. If the command is a *submit* the simulator can infer uniquely the value that the code submits to the object and submits this as input. Different simulators (corresponding to m different codes) may bring in different values to the object agreement. Each of j possible outcomes of the agreement, distributed back to codes, is a value the code may have received in an actual execution.

4. If the command is *receive-reply* a simulator submits a value from the preceding *submit* agreement. At most j distinct values will be submitted to this agreement and one will be chosen.

There are some steps of the simulation that have not been specified. These follow.

1. When does a simulator halt?

2. Which is the next agreement protocol out of those available that a simulator chooses to enter?

A *simulation-rule* is a set of rules (as a function of the local state of a simulator) that resolves the preceding questions for a simulator, making the simulation deterministic.

Lemma A.5 *Any set of halted code views of a simulation constitutes a simplex in the view-graph of the simulated protocol CVG (code view graph).*

Lemma A.6 *Let C^n be an n dimensional complex and let S^n be a subdivided simplex. Let $\mu : (S^n) \rightarrow Bsd(C^n)$ be a simplicial map. Define now a new map $\bar{\mu} : S^n \rightarrow C^n$ such that $\bar{\mu}(v) \in carrier(\mu(v))$ (i.e., $\bar{\mu}(v)$ carries a barycenter to a vertex of the smallest simplex in C^n that contains $\mu(v)$. Then $\bar{\mu}(S^n)$ is the simplicial image of a subdivided simplex.*

Once we fix the simulation rule, the simulation can be viewed as a protocol over the simulators. By [6, 9], the view graph SVG (simulation view graph) of the simulation-protocol is subdivided simplex. If, upon halting, each simulator sees at least one halted code, then the simulation implies a simplicial map from SVG to $bsd(CVG)$. Using lemma A.6 the simulation implies a simplicial map from SVG to CVG. Thus we have the following lemma.

Lemma A.7 *For any simulation rule, if each view of a simulator contains a view of at least one code, then there exists a simplicial map $\bar{\mu}$ from simulators views (SVG) to code views (CVG) such that $\bar{\mu}(SVG)$ is the image of a subdivided simplex.*

The following lemma is crucial for streamlining the necessary condition. It states that every image of a sphere in the code view-graph is exactly the result of a particular simulation rule and particular number of simulators.

Lemma A.8 *Let $\hat{\mu}(S^q) \subseteq CVG$ be the simplicial image in CVG of a subdivided simplex, then there exist a number of simulators and a simulation rule for which there exists a simplicial map $\bar{\mu} : SVG \rightarrow CVG$ such that $\bar{\mu}(SVG) = \hat{\mu}(S^q)$.*

Proof We precede the simulation by a phase that will completely predetermine what will be submitted in each agreement, such that, in principle, no agreement is necessary since all inputs submitted to each given agreement are identical (in the case of *submit* only the values that won are pushed through the agreement. After the submit agreement guarantees than no new value will win, the processor proceeds and submits to the *reply* agreement the value that was returned to that processor independent of whether it won in the *submit* agreement or not.) Thus, the simulation by agreement is an artifact, however simulators go through the agreement steps for reasons to become clear shortly.

Let $q+1$ simulators to do simplex agreement on S^q. Through $\hat{\mu}$ a vertex in S^q is maps to a view in CVG. From this view a simulator determines a unique (since the protocol is full-information) execution prefix (modulo commuting operations) that gives rise to the view. The simulator determines its simulation rule (and what to submit to each agreement) in accordance with the run prefix it possesses. When a code view in $\hat{\mu}(S^q)$ has been produced it halts. Finally, a simulator may determine that an agreement has terminated if, independent of how simulators proceed, the value of the agreement according to previous rules is committed and known. In this case a simulator may simply adopt the committed value as the agreement output. Since we claim that all processors will submit the same value on their tokens, then no matter how many simulators are faulty, the agreement will not be blocked.

Lemma A.9 *If each view in CVG is associated with the shortest prefix run that gives rise to the view then for two views in a simplex one prefix run is a prefix of the other (modulo commuting commuting-operations).*

Proof Since the views lie in a common simplex there exists a run that, modulo commuting operations, has a a prefix each of the prefixes. Since the protocol is full information and reads in snapshot, the prefix property follows. □

The following lemma is the heart of the simulation by agreement. Notice that it "costs" one dead simulator to block any agreement but the *submit*. It takes j dead simulators to block an agreement on (m, j) object, which in turn may block m codes. Finally, with $r(i)$ resiliency, using no more than $r(i) + 1$ simulators such that each one (when not faulty) is completely fair to nonblocked codes we may conclude that if simulator terminates when it is blocked from any remaining agreements, it sees at least one code view.

Lemma A.10 *In a model in which $(i, g(i) + 1)$ narrowing is solvable using a combination of set-consensus objects and the given resiliency, then if we simulate i codes in the model with any rule fair to all i codes, with the halting rule that a simulator terminates when all nonterminated codes have a blocked agreement, then any simulator that halts will have in its view at least one terminal view of a code.*

We are now ready to put it all together. The following lemma proves the necessity of the g-solid conditions 2 and 4.

Lemma A.11 *In a model that achieves $(i, g(i) + 1)$ narrowing by using a combination of set consensus objects and the given resiliency, then if i processors terminate before the others wake up, then any $g(i)$-sphere in their view graph has a fill-in.*

Proof We simulate the i codes. The simulators are given the following rules. They each start at the image of the corner of the $g(i)$-simplex that defines the sphere. The simulators converge on the face they define as long as they see $\leq g(i)$ processors. After convergence they simulate using the simulation rule determined by the convergence. A simulator that sees $g(i) + 1$ other simulators, or gets into a conflict in an agreement, may resort to any fair simulation rule that includes stopping when blocked (i.e., for every code either an agreement is blocked or the code halted). Since when less then $g(i) + 1$ simulators arrive, they produce (depending on interleaving) all and only views on the sphere, and since the simulation under all possible interleaving produces an image of a subdivided simplex, the sphere has a fill-in. This proves condition 2. Condition 4 follows almost verbatim. □

The linking sphere property of the sufficient condition follows from the fact that after some views have been produced, we may consider the simulation as starting from the global state determined by the view. Since views of the remaining processors will lie in the linking sphere of the views that have already been produced we obtain the second property.

The proof is complete.

□