A Simple Algorithmic Characterization of Uniform Solvability *
(Extended Abstract)

Eli Gafni
(eli@cs.ucla.edu)

Abstract

The Herlihy-Shavit (HS) conditions characterizing the solvability of asynchronous tasks over n processors have been a milestone in the development of the theory of distributed computing. Yet, they were of no help when researcher sought algorithms that do not depend on n. To help in this pursuit we investigate the uniform solvability of an infinite uniform sequence of tasks $T_0, T_1, T_2, ...$, where T_i is a task over processors $p_0,\ p_1, ..., p_i$, and T_i extends T_{i-1}. We say that such a sequence is uniformly solvable if there exit protocols to solve each T_i and the protocol for T_i extends the protocol for T_{i-1}. This paper establishes that although each T_i may be solvable, the uniform sequence is not necessarily uniformly solvable. We show this by proposing a novel uniform sequence of solvable tasks and proving that the sequence is not amenable to a uniform solution. We then extend the HS conditions for a task over n processors, to uniform solvability in a natural way. The technique we use to accomplish this is to generalize the alternative algorithmic proof, by Borowsky and Gafni, of the HS conditions, by showing that the infinite uniform sequence of task of Immediate Snapshots is uniformly solvable. A side benefit of the technique is a widely applicable methodology for the development of uniform protocols.

*Work supported by UCLA Academic Senate Grant .

1 Introduction

In a celebrated sequence of results in 1993 and 1994, Herlihy and Shavit completely characterized the solvability of asynchronous tasks for n processors. But, this characterization is not completely helpfull, since in real applications we do not know an upper-bound to the number of processors, or even if we do, we would not like the complexity of the protocol to depend on this bound since in most well designed systems, the potential bound may be orders of magnitude larger than the actual number of processors participating concurrently in an application. Indeed, numerous novel works in recent years are about such contention dependent protocols whose complexity depends only on the actual number of processors that participate [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]. Yet, no general conditions for the existence of such algorithms was previously known.

The observation to make here is that a necessary condition for a contention dependent protocol to exist is that it solves an "infinite" task. We formalize an infinite task to be an infinite sequence of growing *finite* tasks $T_0,\ T_1,\ T_2, ...$, where T_i is a task over processors $p_0,\ p_1, ..., p_i$, and T_i extends T_{i-1}. We call such a sequence of tasks a *uniform* task.

Thus, we would require a sequence of protocols $PT_0,\ PT_1,\ PT_2, ...$, such that PT_i solves T_i, but moreover, when p_i is not in the participating set, then the processors run PT_{i-1}. We call such a protocol uniform protocol. We say that a uniform task is uniformly solvable if there exists a uniform protocol that solves it.

Is uniform solvability different than solvability? Perhaps the sequence is uniformly solvable if each task in it is solvable?

We introduce a novel uniform task for which we are able to show that every constituent task is solvable, yet the uniform task is not uniformly solvable. Our task imposes nontrivial output constraints only on participating sets of the type $p_0,\ p_1,\ p_i,\ p_{i+1},\ \ i = 2, 4, 6, ...$ While uniformity requires that the participating set $p_0,\ p_1$ terminate in a constant time, we show that the time complexity of T_i has to grow with i even when the participating set is only $p_0,\ p_1$. Thus, any protocol for T_1 will eventually fail to be extended.

When is a uniform task uniformly solvable?

The HS necessary and sufficient condition for the solvability of T_i is that there exists a chromatic subdivided simplex $\sigma(s^i)$ (all our tasks are w.l.o.g. inputless, as we argue in the model section) that appropriately maps into complex generated by the output simplexes. We prove that (in hindsight) the natural extension of the HS conditions is necessary and sufficient here too. We just have to consider an "infinite" subdivision instead of finite one. The HS proof method [22] does not extend here since here we deal with an infinite dimension subdivided simplex, and the proof in [22] relies heavily on notions, like a Lebesgue Number, that address solely finite objects. Consequently, we extend the algorithmic proof in [23].

The proof in [23] shows, via direct implementation, the equivalence between the read-write (RW) Snapshots model of computation and the novel model of Iterated Immediate Snapshots (IIS). The IIS model produces a protocol complex which is a subdivided simplex. Thus we extend this equivalence to the uniform computation domain. While the implementation of Snapshot by IIS extends almost verbatim (the only change is that we have to resort to 'non-uniform' IIS [24], rather than IIS), the significant part is showing a uniform implementation of IIS by Snapshots. The original implementation in [25] is not uniform. A uniform algorithm appeared first in an precursor draft of this paper [26] and later in [13]. Here we give a new algorithm of O(n^2) complexity (in the RW model). An improvement over the algorithm of O(n^3) in [13]. This is significant as many algorithms use IS as a subroutine.

Finally, we give a sufficient condition and a corresponding methodology of deriving uniform protocol for a problem by solving a certain finite task which is a variation of the original task. In particular we notice that Simplex Convergence Problem [22] satisfies these sufficient conditions, which is an alternative proof of the sufficiency part of our characterization. These sufficient conditions hold for all natural problems we know.

Thus the contribution of this paper is: 1. We define the notions of Uniform Task and Uniform Protocol. 2. We show that solvability of each finite task in a uniform task does not imply the uniform solvability of the uniform task. 3. We give necessary and sufficient conditions for uniform solvability. 4. We give a methodology that works in many cases to reduce uniform solvability to a problem about standard solvability, and, 5. We give a new $O(k^2)$ algorithm for Immediate Snapshots, where k is the cardinality of the participating set.

The paper is organized as follows: Section 2 introduces some facts about algebraic topology to make the paper self contained. Next, in section 3, we introduce the model section, followed in section 4 by our example of a uniform task that is not uniformly solvable. Next, in section 5, we establish uniform solvability conditions in the Iterated Immediate Snapshots mode. We then show that these conditions apply for all R/W models, by showing in section 6, that the Snapshots and the Iterated Immediate Snapshots models uniformly implement each other. We then discuss the sufficient conditions and a methodology to design uniform protocols, in section 7, and we conclude, in section 8, with open problems.

2 Topological Preliminaries

An n-dimensional simplex is a set of $n+1$ vertices. A simplicial complex is a set of simplexes closed under intersection and containment. A complex C^n is pure of dimension n if every simplex is a subset of an n-dimensional simplex, and if the n-dimensional simplexes are only subset of themselves.

A complex C has an embedding in a Euclidean space of high-enough dimension such that vertices in a simplex are affinely independent. All our complexes will be embedded.

A complex $B(A)$ is a subdivision of a complex A if there is an embedding of both in the same space such that: 1. The convex hull of any simplex of $B(A)$ is contained in the convex hull of a simplex of A and, 2. The union of the convex hull of simplexes in $B(A)$ is the same as A. If s is a simplex of $B(A)$, then $carrier(s, A)$ is the unique smallest simplex of A whose convex hull contains the convex hull of s. For a complex $C \subseteq B(A)$, $carrier(C, A) = \cup_{s \in C}\ carrier(s, A)$.

A map of vertices of complex A to vertices of complex B is simplicial if any simplex in A maps to a simplex in B. The map is dimension preserving if the image of any simplex has the same dimension as that of the source. A coloring of a complex C^n is a dimension-preserving simplicial map $\mathcal{X}$ from C^n to some simplex s^n of dimension n. A complex together with coloring is a chromatic complex. A simplicial map μ from a chromatic complex A to B is *color preserving* if for all $v \in A,\ \mathcal{X}(v) = \mathcal{X}(\mu(v))$. A complex A is a subcomplex of B if every simplex in A is a simplex of B. If B colored, then A inherits the colors from B. For $C \subseteq A$, where A is chromatic, $\mathcal{X}(C) = \cup_{v \in C} \mathcal{X}(v)$. If $A(C)$ and $B(C)$ are subdivisions of C then a simplicial map μ from $A(C)$ to $B(C)$ is carrier preserving, if for all $v \in A(C),\ carrier(v, C) = carrier(\mu(v), C)$. the source simplex.

A subdivided simplex $A(s^n)$ is a complex that is a subdivision of some simplex s^n. A face of A corresponding to $s^q \subseteq s^n$, denoted by $A(s^q)$, is the subcomplex of the simplexes of A whose carrier is a subset of s^q. The complex of the $n-1$ dimensional faces of a subdivided simplex

$A(s^n)$, called $boundary(A(s^n))$, is an $(n-1)$-dimension *sphere*. Let μ be a simplicial map from $boundary(A(s^n))$ to C. We say that $\mu(boundary(A(s^n)))$ has a fill-in, if there exists $A'(s^n)$ and a simplicial map μ' from $A'(s^n)$ to C, such that $boundary(A'(s^n)) = boundary(A(s^n))$ and μ' agrees with μ on the boundary. The simplicial image in C of $(n-1)$-sphere has a fill-in if there exist a simplicial map from the subdivided simplex to C that extends the map of the sphere.

A complex C has no hole of dimension k if every image of a $(k-1)$-sphere in C has a fill-in. For $s^q \in C$, $star(s^q, C)$ is union of the convex hull of all the simplexes in C that contain s^q. $Link(s^q, C)$ is the union of the convex hull of all the simplexes in $star(s^q, C)$ that do not contain any vertex of s^q. The first barycentric subdivision of an embedding of s^n, denoted $Bsd(s^n)$, is obtained recursively, by planting a vertex in the center of s^q and considering it a simplex when added to any simplex of the recursively barycentrically subdivided $n-1$ faces of s^n, where the subdivision of the zero-dimension simplex is itself. The $Bsd(C^n)$ of a complex C^n is the subdivision of C^n obtained by replacing each $s^q \in C^n$ with $Bsd(s^q)$. $Bsd^k(C) = Bsd^{k-1}(Bsd(C))$, where $Bsd^0(C) = C$.

Unfortunately there is no coloring of $Bsd^k(s^n),\ k > 0$ which is dimension preserving. To achieve this, we define the Standard Chromatic Subdivision $SCS(s^n)$. Intuitively we view each vertex and edge introduced by the subdivision as made of 'rubber' that is 'pulled' by the subdivision vertices of the lower dimension faces. Formally, $SCS(s^n)$ is obtained from $SCS(s^{n-1})$ as follows: Inductively chromatically subdivide each face of s^n, then add a vertex of color $\mathcal{X}(v)$ across from the $n-1$-dimension face that does not in contain v, and connect it to all the vertices on the face. Then connect all the added nodes, and consider every complete subgraph as a simplex. By planting the new nodes close enough to their corresponding face the embedding is easily accomplished. Similarly, as with Bsd we define $SCS^k(s^n)$ to be $SCS(SCS^{k-1}(s^n))$, with $SCS^0(s^n) = s^n$.

We can also define a variation of Bsd and SCS of s^n with respect to a set of *stopped* simplex $s^q \subseteq s^n$ by subdividing $s^n - s^q$ and then coning off the vertices of the subdivision with the vertices of s^q. We notice that in constructing $SCS^k(s^n)$ we can now specify which vertices we stop in going from $SCS^i(s^n)$ to $SCS(SCS^i(s^n))$. In this paper, once a vertex its stopped it is stopped in all subsequent level of the inductive construction.

3 Models

3.1 SWMR Atomic Snapshot Memory Model and the Full-information Protocol

In the SWMR atomic snapshot memory model [27], each processor p_i, $i = 0, 1, ...,$, has a cell C_i (initially $\perp$) that is exclusively written by it and is read by all processors. Each processor alternates between writing its cell and reading all cells in a single atomic snapshot.

Since we deal with infinite number of processors, we can assume w.l.o.g. that each processor has a single input. To model a processor with few inputs we just consider each input to belong to a distinct processor.

An execution of the full-information protocol in the model is an infinite sequence of processor IDs where the first appearance of a processor is interpreted as write, the second as snapshot, and so on. Given an execution of the full-information protocol, the set of the processors which appear at least once is called the participating set. In its first write, a processor p_i writes its ID to C_i. In subsequent writes, it writes the precise encoding of the whole memory it last read in a snapshot. This last snapshot and whether p_i has written this last snapshot back to C_i is p_i's local state.

Given an infinite sequence of integers $k_0, k_1, k_2, ...,$ where all the k_is but finitely many of them are 0, we may define the complex of the full-information protocol corresponding to an execution prefix in which each processor p_i has gone through k_i rounds of writes and snapshots. The vertices of the complex are all the possible pairs $(p_i, view_i)$ for all i such that $k_i > 0$, where p_i is a processor ID and $view_i$ is the local state of p_i after k_i rounds in any of the prefixes above. A set of vertices form a simplex when their associated local states are consistent with a single execution prefix.

We assume that processors IDs are identified with the vertices of a simplex s^∞ and thus, since a processor has a single view in an execution prefix, the complex is chromatic.

3.2 Tasks and Uniform Tasks

A task over processors $p_0, p_1, ..., p_n$ is a processor-set/output relation. For each subset of the $n+1$ processors, a task specifies all the output tuples that may result in executions where this set of processors is the participating set.

For instance, in the $(n+1, k)$ set-consensus task [28], each processor has its ID as an input and, for each participating set, each processor decides on an ID from the set, such that the number of distinct IDs decided on does not exceed k.

Tasks over $n+1$ processors can be represented by a chromatic simplicial complex, O^n, and a map Δ [29]. O^n is a complex whose vertices are all pairs $(p_i, DecisionValue_i)$, such that the task allows p_i to output $DecisionValue_i$. A set of vertices is a simplex if there is an output tuple containing them. The map Δ is a point-to-set mapping from s^n to O^n such that if $s_i \in s^n$, $s_o \in O^n$, and $s_o \in \Delta(s_i)$, then $\mathcal{X}(s_i)$=$\mathcal{X}(s_o)$, where the color of a vertex is the name of the associated processor, and $\Delta(s_i)$ are all the tuples that the processors in the participating set corresponding to s_i are allowed to halt with.

A uniform task T is a sequence of task $T_0, T_1, T_2, ...$ over processor sets $\{p_0\}$, $\{p_0, p_1\}$, etc. respectively, such that:

1. For all $s^q \subset s^i$ that does not include the vertex that corresponds to p_i, $\Delta_{T_i}(s^q) = \Delta_{T_{i-1}}(s^q)$, and
2. for $s_o \in \Delta_{T_i}(s^q)$ there exist $s_O \in \Delta_{T_i}(s^q \cup \{v(p_i)\}$, such that $s_o \subset s_O$.

For example the following is not a uniform task: In the task T_i, processor p_j in a solo execution outputs the jth vertex of an embedded unit simplex, while for a participating set P of size $|P|$ each processor outputs a point in the embedded face that corresponds to P such that the Euclidean distance between any pair of output points is at most $1/|P|$. Although each of the tasks T_i is solvable, they fail on the second condition. Task T_i can be uniformized to comply with the second condition. But if we proceed ad infinitum it leaves only the distance 0 to survive, which makes each T_i unsolvable.

3.3 Wait-free and Uniform Solvability

A partial map from local states of processors in the full information protocol to values is a decision function, or a protocol, if in any execution after a local state of a processor is mapped to a value, all subsequent local states of the processor are mapped and to the same value. A processor's appearance in an infinite execution is (or has) decided if the corresponding local state in the execution is mapped to a value. A processor is decided in an execution if it has an appearance in the execution which is decided. A task is wait-free solvable in the atomic snapshot model if, there exists a protocol such that for any infinite execution of the full-information protocol, with participating set P, all processors that appear infinitely often are decided, and the output tuple ${s'}_o$ that results in from the decisions in the execution can be extended to an output simplex $s_o \in O^n$, ${s'}_o \subseteq s_o$ and $s_o \in \Delta(s^P)$.

Let T be a task that is wait-free solvable. A task is bounded wait-free solvable if there exists a bound b such that in any execution, any processor that appears b times in the sequence has decided. In [23] it is proved that if T_n is wait-free solvable, then it is bounded wait-free solvable.

A uniform task is uniformly wait-free solvable if:

1. The decision function for the protocol for T_i extends the one for T_{i-1}, and
2. In an execution in which the participating set is finite (i.e. we are in the 'finite arrival' model cf [18]), a participating processors that appears infinitely often decides.

3.4 Immediate Snapshot Model

The immediate snapshot model introduced in [30, 31] is a restriction of the atomic snapshot model in the sense that its set of executions is a subset of the atomic snapshot model. It comprises executions in the atomic snapshot model in which each maximal run of writes is followed by a maximal run of snapshots by the same processors. Consequently, we can condense the odd-followed-by-even appearance of a processor as a single operation called WriteRead($value$). Thus, an execution is a sequence of sets of processors. In [25] it was proven that immediate snapshots model over n processors can be simulated by the atomic snapshot model and therefore cannot solve anything not solvable in the atomic snapshot model.

3.5 Iterated Immediate Snapshot (IIS) Model

A one-shot immediate snapshot is an immediate snapshot model that allows each processor to WriteRead only once. In the iterated immediate snapshot model, we have a sequence of one-shot immediate snapshot memories, $M_0, M_1, ...$ The full-information protocol execution starts by p_i WriteReading its input to M_0. It then applies the output from M_i, $i > 0$, as an input value to M_{i+1}, ad infinitum.

Formally, one-shot immediate snapshot is specified as follows. For any input val_i to p_i (where w.l.o.g. $\forall i, j,\ val_i \neq val_j$ by adding processor's Id to the input), if the set of participating processors is P, then p_i's output is a set S_i of inputs to processors, such that:

1. $val_i \in S_i, \forall p_i \in P$
2. $S_i \subseteq S_j$ or $S_j \subseteq S_i, \forall p_i, p_j \in P$
3. $val_i \in S_j \Rightarrow S_i \subseteq S_j,\ \forall p_i, p_j \in P$

When $val_i \in S_j$ we say that p_j *sees* p_i. A full-information execution in the IIS model is a two dimensional infinite array $L[\infty, \infty]$, where $L(i, j)$ is the state of p_i after going through memory M_j. Inductively, if the local state of processor p_i after its appearance in M_j is v_{ij}, then its local state after its $j+1$ appearance is the result of inputing (p_i, v_{ij}) to a one-shot immediate snapshot M_{j+1}. Many sets are now called participating in an execution described by L. We say that a finite set of processors P is participating if in no memory M_j in the sequence, a processor $p_r \in P$ 'sees' processor $p_s \notin P$, i.e. a participating set is closed under the relation 'sees'.

A task T_n is wait-free solvable in the iterated immediate snapshot model if for all participating sets which are subset of $\{p_0, ..., p_n\}$ and for p_i, there is a j_i large enough, such that the output of p_i from M_{j_i} can be mapped to a decision-value satisfying the task. It is easy to see that wait-free solvability here also implies bounded wait-free solvability. The notion of uniform solvability as before requires that a decision state in a protocol for T_{i-1} is a decision state for the protocol of T_i, with the same value. Moreover any processor in an infinite execution, that belongs to a finite participating set must eventually decide.

4 A Uniform Task which is not Uniformly Solvable

In this section we explicitly describe a uniform task $T = T_0, T_1, ...$ and prove that it is not uniformly solvable. Let $e(n) = n - 1$, for odd n, and $n - 2$ for n even. The task T_n imposes non-trivial constrains only on subsets of participating sets $\{p_0, p_1, p_i, p_{i+1}\}$ $i = 2, 4, ..., e(n)$. We define $\Delta(\{p_0, p_1, p_{e(i)}, p_{e(i)+1}\}$ to be $SDS^i(\{p_0, p_1, p_{e(i)}, p_{e(i)+1}\} = s^3)$ where on the face $(p_0, p1)$ rather than returning the vertices themselves the output for processors p_0 and p_1 is defined by mapping the path from the p_0 color corner to the p_1 color corner snakewise onto the path $a--b--c--d, 3^i$ times (see figure 1).

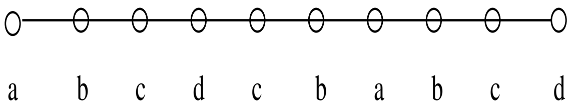

Figure 1. Snake-wise map onto $a - b - c - d$.

Proposition 4.1 *For all n the task T_n is solvable.*

Proof Processors solve simplex convergence on $SDS^{e(n)}(s^3)$. Processors on the face (p_0, p_1) output a, b, c, or d, according to the snake mapping above. For a participating set that include either or both $p_{e(i)}$, $p_{e(i)+1}$, vertices in $SDS^{e(n)}(s^3)$, which are not on the face (p_o, p_1) map to $SDS^{e(i)}(s^3)$, by considering the position of the vertex going down the recursion just the last i levels. An example of the mapping from $SDS^2(s^2)$ to $SDS^1(s^2)$ is shown in figure 2. $\square$

It is obvious that sequence T of tasks $T_0, T_1, ...$ complies with the requirements to constitute a uniform task.

Proposition 4.2 *The uniform task T is not uniformly solvable.*

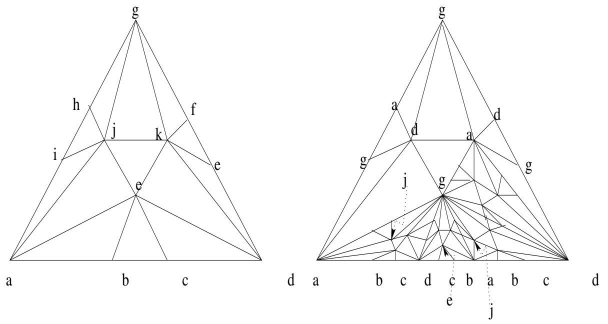

Figure 2. figure=Map of $SDS^2(s^2)$ to $SDS^1(s^2)$.

The observation is that T_n is solvable since we simplex converge on $SDS^{e(n)}(s^3)$, which requires at least $e(n)$ steps [24]. To be solved uniformly in an execution with the participating set $\{p_0, p_1\}$ the processors have to decide in constant amount of time. But then by waking up $p_{e(i)}$, $p_{e(i)+1}$ for i large enough, if p_0 ,say , decided c we do not 'know' which c among all the nodes that map to c it meant. That this disagreement between $p_{e(i)}$ and $p_{e(i)+1}$ cannot be reconciled in any model equivalent to the RW model is the consequence of FLP [32].

Proof Formally, consider all the runs in which p_0 and p_1 terminate before $p_{e(n)}$ or $p_{e(n)+1}$ start. By the HS conditions we generate a 3-connected complex which is a subcomplex of $SDS^{e(n)}(S^3)$, and in which the face corresponding to p_0 and p_1 has been mapped snakewise to $a - b - c - d$. By taking n large enough we can get two runs in which the final local state of, say, p_0 is mapped to c for different continuations by $p_{e(n)}$ and $p_{e(n)+1}$, such that their final local states maps to distinct values in the two runs.

We notice that no two simplexes in which processor p_0 (or p_1) output the same letter share a vertex that is an output of $p_{e(i)}$ or $p_{e(n)+1}$. Thus there is a gap in the outputs of $p_{e(n)}$ and $p_{e(n)+1}$ when they start with p_0's final state as above. We can now solve consensus. We accomplish this by taking two processors, ps_0 that BG simulates [33] one continuation by $p_{e(i)}$ and $p_{e(n)+1}$, and ps_1, that BG simulates the other. Thus ps_0 and ps_1 can reach consensus by determining on which side of the gap they ended up in. Contradiction. $\square$

5 Emulations and Characterization

5.1 Characterization of the Iterated Immediate Snapshots Full-Information Protocol

Let $V = \{(p_i, S_i) | i = 0, ..., n, S_i \subseteq \{p_0, ..., p_n\}, p_i \in S_i\}$ be all the possible tuples where S_i is an output of p_i in the immediate snapshot model when each processor inputs its ID and the participating sets are subsets of $\{p_0, ..., p_n\}$. Fix the protocols for $T_0, T_1, ...$

Elsewhere [22] it was argued that the protocol complex for the participating set $p_0, ..., p_n$ after k steps of IIS is isomorphic to $SDS^k(s^n)$. We notice [24] that if we 'stop' a local state after it maps to a decision by the protocol we get is still a subdivided simplex. Thus, since T_i is solvable in at most some k_i iterated immediate snapshot steps, then if we have a uniform protocol, when solving T_{i+1} in $k_{i+1} > k_i$ iterated immediate snapshot steps, all the vertices on the $(p_0, ..., p_i)$ face could be stopped by the k_i step. Thus the solution to T_{i+1} builds a chromatic subdivided simplex which agrees with the subdivided simplex built by the solution to T_i. Obviously the existence of such a sequence of subdivided simplexes that maps to the output is necessary and sufficient.

Consider an infinite sequence of non-decreasing integers $0 \leq k_0 \leq k_1 \leq k_2 \leq ....$ For a prefix $0 \leq k_0 \leq k_1 \leq ... \leq k_n$ we define a chromatic subdivided simplex A_n as 'stopped' $SDS^{k_1,...,k_n}(s^n)$ where after $k_i, i = 0, 1, ..., n$ steps we stop vertices of face $(p_0, .., p_i)$ respectively. Obviously A_n extends A_{n-1}.

Proposition 5.1 *A uniform task T is wait-free solvable in the iterated immediate snapshot model iff there exits a sequence $0 < k_0 < k_1, < k_2, ...,$ a sequence of subdivided simplexes $A_0, A_1, A_2, ..,$ as above, and a sequence of color-preserving simplicial map $\mu_i : A_i \to O^i$, such that $\forall s \in A_i$, $\mu_i(s) \in \Delta(carrier(s, A_i))$, and μ_i agrees with μ_{i-1} on the face $(p_0, ..., p_{i-1})$ of A_i.*

6 Uniform Emulations

In this section we show the uniform equivalence of the models of Atomic Snapshots and IIS. The notion of uniform solvability extends naturally to the notion of uniform emulation, and we do not need elaborate on it.

6.1 Uniform Emulation of Atomic Snapshot Memory by Iterated Immediate Snapshot Memory

This subsection follows verbatim the emulation in [23]. We repeat it here to make the paper self-contained.

W.l.o.g we may assume that a protocol Π in SWMR atomic snapshot memory proceeds by each processor p_i executing a $WriteRead_i(v)$ operation in which it writes v to C_i and then returns a snapshot all in a single operation.

To show that a given model M can emulate an atomic-snapshot full-information protocol we may assume w.l.o.g. that the value v that p_i writes in its jth $WriteRead_i(v)$ operation is just j. Thus an operation returns an n-vector of integers (including 0). M emulates correctly if we can argue that the vectors returned satisfy:

1. In the vector returned by the operation $WriteRead_i(k)$, $v_i = k$.
2. Any two vectors returned are either the same or one majorizes the other.
3. No value j for p_i is returned in an execution in which p_i has not $WriteRead$ j times.

To emulate $WriteRead_i(j)$, we assume inductively that processor p_i just finished emulating $WriteRead_i(j-1)$ and returned a vector of integers $(v_1, ..., v_n)$ between 0 and k. The initial vector for p_i is 0 everywhere.

Processor p_i increment v_i to j and submits this vector to the next stage of the immediate snapshot. We call this vector its "estimate" of the values of cells. In the immediate snapshot it obtains a set of estimate vectors. Its update its estimate by taking the maximum entry-wise on all these vectors.

It repeats submitting its estimate vector to the next stage of the iterated immediate snapshot until all estimate vectors returned to it by the immediate snapshot are the same. It then finished the operation $WriteRead_i(j)$. It returns the estimate vector of the stage as the output for the operation and proceeds to emulate $WriteRead_i(j+1)$.

The correctness of the algorithm hinges on two simple facts: 1. Only a single estimate vector may be adopted as an output at a stage, and all processors returning that vector have the correct required value for their own entry. 2. By The virtue of estimates being entry-wise maximum, if a vector is returned as output in one stage, all estimate vectors input to the next stage majorize it.

The liveness of the algorithm follows if the participating set is finite, since if in a stage no processor outputs a value then there must be a processor that strictly increased its estimate. Since processors do not increase their own value without obtaining an output, the number of possible estimate increase is bounded.

We notice in passing that the liveness will fail if the participating set is not finite, and consequently the emulation fails for what is called in [18], the infinite arrival model, which requires outputs in an execution, even in the face of unbounded participating set, as long as a processor took unbounded number of steps.

Once we have the emulation, we have established that the conditions for wait-free solvability of the Atomic Snapshots model are sufficient conditions for solvability in the Iterated Immediate Snapshots model.

6.2 Uniform Emulation of Immediate Snapshot In the SWMR Model

Since here we are also interested in efficiency our model of computation will be low-level, rather than the strong Snapshot model. Obviously if we can uniformly implement IIS by the SWMR model, we can do the same in the Snapshots model. In the SWMR (Single-Writer- Multi-Reader) model of computation for processors $p_0, ..., p_n$, we have $n+1$ cells $C_0, ..., C_n$ where cell C_i is exclusively written by p_i and read by all. Processors alternate between writing their own cell, and reading all other cells asynchronously. We will assume an oracle that tells a processor reading C_j whether all subsequent cells $C_{j+1}, C_{j+2}, ...,$ all hold the initial reserved value $\perp$, or not (obviously the infinite processor Snapshot model and the IIS model have this oracle implicit). To establish the existence of a uniform protocol for the uniform immediate snapshot tasks $IS = IS_0,\ IS_1, ...$ it is enough to show that if we have a protocol for $IS_{n/2}$ we can obtain a protocol for IS_n which agrees with the former protocol on the participating sets from $\{p_0, ..., p_{n/2}\}$.

The idea is as follows: Processors post (register) their participation. Processor $p_i,\ i \leq n/2$ executes the protocol for $IS_{n/2}$. When it terminates, it posts its output S_i for $IS_{n/2}$. It then observes whether only processors from $\{p_0, ..., p_{n/2}\}$ registered. If positive it quits, returning S_i. Otherwise, it joins the protocol for IS_n suspecting that all processors in $\{p_0, ..., p_{n/2}\} - S_i$ quit after terminating $IS_{n/2}$. Similarly processor $p_i,\ i > n/2$ post their name and then observe outputs in $IS_{n/2}$, and proceed with the largest S_j posted to solve IS_n. The output produced by processors participating in the protocol for IS_n should be compatible with the output posted by processors that departed in $IS_{n/2}$.

The observation to make is that a processor that quits in $IS_{n/2}$ is guaranteed that its output has been observed by all processors $p_i,\ i > n/2$ that proceed solve IS_n. Thus we need here to solve a variation of the original task IS_n, which we call $BC - IS_n$ (BC for Backward Compatible). Each processor wakes up with an input which is an output subsimplex, where all the input come from the same simplex, some possible outputs, all sets mutually coming from a single output tuple, and, if all participating set agree on some output for some processor, the tuple they produce is such that it can be extended by the by the input subsimplex they all agree on to an output in IS_n (for the participating set in IS_n, which is the participating set in $BC - IS_n$, together with all the processors that appeared in the input sets of $BC - IS_n$).

The algorithm to solve $BC - IS_n$ is a variation of the algorithm for IS_n in [23, 34]. While there, each processor p_i carried only it own ID, here processor p_i carries it own and others (S_i). We make processors decent $n+1$ *levels*, from level l_{n+1} to level l_1. Each level l_k consists of two sub-levels, $sl_{k,2}$ on top of $sl_{k,1}$. Each processor p_i is associated with a fixed pair (p_i, S_i), where for $i \leq n/2$, S_i is the immediate snapshot returned by p_i in $IS_{n/2}$, while for $i > n/2$, S_i is the largest snapshot p_i has seen posted for $IS_{n/2}$. The invariant we are going to maintain is that no more than k processors descent to level l_k.

Sub-level $sl_{k,2}$ is responsible to maintain the invariant. Sub-level $sl_{k,1}$ is responsible to signal to p_i that although it may want to stop at $sl_{k,2}$ some other processor may have 'dragged it down' already, and therefore it should not stop. Stopped processors at level l_k return the IDs that descended to level k. Thus if p_i stops at level l_k, its ID should not be carried to level $k-1$, by some other processor.

At sub-level $sl_{k,2}$ processor p_i posts an ordered-pair (p_i, S_i). For (p_i, S_i) let U_i be $U_i = \{p_i\} \cup S_i$.

1: Processor p_i reads $sl_{k,2}$:

1.1: $|\cup_j U_j| < k$ where U_j appears is $sl_{k,2}$
proceed down to sub-level $sl_{k,1}$,

1.2: $|\cup_j U_j| = k$ where U_j appears is $sl_{k,2}$

1.2.1 $p_i \notin S_i$

1.2.1.1 If there exists (p_j, S_j) in $sl_{k,1}$ with $p_i \in S_j$, proceed to $sl_{k,1}$

1.2.1.2 else, return $\cup_j U_j$

1.2.2 $p_i \in S_i$,

1.2.2.1 $|S_i| = k$, return $\cup_j U_j$

1.2.2.2 else, if for all $p_j, p_j \notin S_j$ we have $p_i \in S_j$, or
there exists (p_j, S_j) in $sl_{k,1}$ with $p_i \in S_j$, then
proceed down to sub-level $sl_{k,1}$,

1.2.2.3 else, return $\cup_j U_j$

2. In $sl_{i,1}$ processor p_i posts its pair (p_i, S_i), and reads $sl_{i,2}$ again. It (internally) sets $S_i := S_i \cap \{S_j | p_j \notin \S_j\}$, and proceeds with the new (p_i, S_i) to post it at $sl_{k-1,2}$.

Sketch of a correctness proof: We now prove the invariant. Inductively, $|\cup_j U_j|$ at sub-level $sl_{k,2}$ is less equal k. To see that this inductively holds for $sl_{k-1,2}$, we notice that a processor does not stop in three cases: 1. It observes (1.1) $|\cup_j U_j| < k$, thus the number of ID's that may observe this is at most $k-1$ (by considering the first processor p_i that causes $|\cup_j U_j| = k$. This processor and subsequent ones will not observe $|\cup_j U_j| < k$, 2. It sees p_i in some S_j at $sl_{k,1}$ (1.2.2.1) – but then processor p_j inductively observed $|\cup_j U_j| < k$, and since necessarily $S_i \subseteq S_j$ (since $p_i \in S_j$ and S_i is the smallest immediate snapshot to contain p_i), and since p_j proceeded down we do not violate the invariant. 3. All processors that do not carry their own IS carry p_j (1.2.1), –since inductively what they carry is an IS, everybody is compatible yet with him returning it own IS.

But the invariant does not guarantee correctness, since we have to show if $p_i,\ i \leq n/2$ stops at level k no processor p_j at level $k-1$ is such that $p_i \in S_j$. If p_i stops at $sl_{k,2}$, it did not see a p_i in any S_j at $sl_{k,1}$, thus all we have to guarantee is that a processor which later tries to 'drag' p_i and will not post S_j that contains p_i at level $k-1$. But

Protocol for Processor i:
Initially:
Arrays $p_{SL_{k,j}}[0,...,n],\ S_{SL_{k,j}}[0,...,n]$
$k = 1,...,n+1,\ j = 1,2$ of set of numbers
initially empty.
Array $S[0,...,n]$ of set of numbers, initially empty

$level(k,2)$
$p_{SL_{k,2}}[i] := i,\ S_{SL_{k,2}}[i] := S[i]$
$U := A := B := \emptyset$
for $j = 0$ to n do
$U := U \cup p_{SL_{k,2}}[j] \cup S_{SL_{k,2}}[j]$ od
if $|U| < k$ call $level(k,1)$
if $|U| = k$ and $i \notin S[i]$ then do
for $j = 0$ to n do
if $i \in S_{SL_{k,1}}[j]$ then call $level(k,1)$ od
return U od
if $|U| = k$ and $i \in S[i]$ then do
if $|S_i| = k$, return U, else do
for $j = 0$ to n do
if $(S_{SL_{k,2}}[j] \neq \emptyset,\ or\ \ j \notin S_{SL_{k,2}}[j])$ then do
$A := A \cap S_{SL_{k,2}}[j]$ od
$B := B \cup S_{SL_{k,1}}[j]$ od
if $i \in A \cup B$ then call $level(k,1)$,else return U, od,od

$level(k,1)$
$p_{SL_{k,1}}[i] := i,\ S_{SL_{k,1}}[i] := S[i]$
for $j = 0$ to n do
if $j \notin S_{SL_{k,2}}[j]$ then
$S[i] := S[i] \cap S_{SL_{k,2}}[j]$ od
call $level(k-1,2)$

Figure 3. Code for $BC - IS_n$

if p_i stopped at level k with $|S_i| < k$ it is because some processor p_m such that $p_m \notin S_m$ at $sl_{k,2}$ does not carry p_i. Now, processor p_j will intersect its S_j with S_m and will lose p_i for the next level.

Obviously, the complexity of the algorithm is $O(n^2)$ since there are $O(n)$ operations at a level and there are $n+1$ levels. Elsewhere, it is shown that by employing MWMR variables we can get that if processor (conceptually renamed) p_i participates then the size of the participating set is O(i).

7 Sufficient Condition for Uniform Solvability

Generalizing the idea of the uniform algorithm for IS above, we derive a sufficient condition for the existence of uniform solution for the uniform task $T = T_0, T_1, ..., T_n, ...$ For an input-less task T_n we define the multi-input task $BC - T_n$. Then, if for all n $BC - T_n$ is solvable the uniform task T is uniformly solvable.

In the task $BC - T_n$ processor p_i wakes up with a partial output tuple o_{p_i} such that there exists an output tuple o where for all i, $o_{p_i} \subseteq o$. Let $o_{agree} = \cap_j o_{p_j}$. Then to solve $BC - T_n$ processors' output should be such that together with o_{agree} it can be completed to a valid output tuple.

The strategy of solving T is for processor p_i to solve $BC - T_k,\ k = 0, 1, 2, 3, ...$ until an output is obtained. It posts its arrival, scans $BC - T_0, ..., BC - T_{i-1}$ for the latest output it sees for processors that posted outputs (think that each processor has a dedicated place in memory in which it writes new tentative output for itself overwriting previous ones), and takes these outputs as its input to $BC - T_i$. Terminating in $BC - T_i$, it posts its tentative output and checks whether a processor $p_j,\ j > i$ registered. If does, it scans for the latest outputs and proceeds to $BC - T_{i+1}$, etc.

The correctness of the scheme is obvious.

To see why the conditions are only sufficient but not necessary, notice that our counter example task at section 4 has the property that $BC - T_n$ for large enough n is not solvable, since, again, if processors $p_{e(n)}$ and $p_{e(n)+1}$ wake up with some values from a,b,c,d for p_0 and p_1 they cannot resolve which specific vertices in the protocol complex these letters map to. We can make the task uniform by letting $T_m,\ m > n$ be exactly T_n with no additional constrains added. Obviously we got a uniformly solvable uniform task that does comply with our conditions, thus they are not necessary.

We now apply this strategy to the uniform simplex convergence problem.

Let $SIM = SIM_0, SIM_1, ...$ be the uniform simplex convergence task where SIM_n is the task for processors to converge on a given subdivided chromatic simplex

$SIM(s^n)$ [22]. I.e for participating set P corresponding to the simplex s^P, $\Delta(s^P) = SIM(s^P)$.

That $BC-SIM_n$ is solvable is a simple corollary of our simplex agreement algorithm in [23, 34]. There also, as the algorithm progress some processors are suspected to have converged and the rest of the convergence proceeds in the link of the converged processors.

Since now we have noticed we can converge uniformly on any infinite dimensional chromatic subdivided simplex, we can amend our necessary and sufficient conditions for uniform solvability. We do not need any more a mapping from a chromatic subdivided simplex that is built by stopped IS iterations. Any chromatic subdivided simplex will do. Thus in a way it is the necessity part of the characterization which is hard.

Thus we get the necessary and sufficient conditions:

Proposition 7.1 *A uniform task T is R/W wait-free uniformly solvable iff there exits a sequence chromatic of subdivided simplexes $\sigma(s^0), \sigma(s^1), ...$ such that $\sigma(s^{i-1})$ is the $i-1$-dimensional face of $\sigma(s^i)$ across from p_i, and a sequence of color-preserving simplicial map $\mu_i : \sigma(s^i) \to O^i$, such that $\forall s \in \sigma(s^i)$, $\mu_i(s) \in \Delta(carrier(s, A_i))$, and μ_i agrees with μ_{i-1} on the face $(p_0, ..., p_{i-1})$ of $\sigma(s^i)$.*

8 Conclusion

We have defined Uniform Protocols independent of complexity measures (e,g. a protocol is uniform if it complexity is independent of n) allowing us to characterize them. The characterization helps in resolving questions whether a uniform task is uniformly solvable. Moreover, the way we derive the characterization provides a methodology to develop uniform protocols. All the uniform algorithms published todate have an alternative protocol of better or equal complexity derived by our methodology. An important open problem is nevertheless to extend the characterization to the infinite arrival model [18]. This will be interesting since assuming long enough 'idle periods' of no arrivals is not realistic in certain internet applications. It is also discouraging that we may need to wait for an 'idle period' in order to be able to obtain an output. In [18] it is shown that many natural tasks are solvable even in the infinite arrival model. Characterizing these task will truly characterize "wait-freedom."

Acknowledgment: I would like to acknowledge the contribution of Elias Koutsoupias to this paper. Elias coauthored with me an unpublished precursor to this paper [26], which contained the first uniform protocol for Immediate Snapshots and an 'algebraic' proof and presentation of a uniform task which is not uniformly solvable.

References

[1] J. Aspnes, G. Shah, and J. Shah, "Wait-free consensus with infinite arrivals," *To appear in STOC02*, 2002.

[2] H. Attiya and A. Fouren, "Adaptive and efficient algorithms for lattice agreement and renaming," *SICOMP*, to appear. An extended abstract appeared in [11].

[3] H. Attiya and A. Fouren, "Algorithms adaptive to point contention," *JACM*, 2001. submitted for publication.

[4] Y. Afek, H. Attiya, A. Fouren, G. Stupp, and D. Touitou, "Long-lived renaming made adaptive," in *PODC99*, pp. 91–103, 1999.

[5] Y. Afek, H. Attiya, A. Fouren, G. Stupp, and D. Touitou, "Adaptive long-lived renaming using bounded memory." www.cs.technion.ac.il/~hagit/pubs/AAFST99disc.ps.gz, Apr. 1999.

[6] Y. Afek, G. Stupp, and D. Touitou, "Long-lived and adaptive collect with applications," in *FOCS99*, pp. 262–272, 1999.

[7] Y. Afek, G. Stupp, and D. Touitou, "Long-lived adaptive splitter and applications." Unpublished manuscript, Dec. 1999.

[8] Y. Afek, G. Stupp, and D. Touitou, "Long-lived and adaptive atomic snapshot and immediate snapshot," in *PODC00*, pp. 71–80, 2000.

[9] J. Anderson and Y.-J. Kim, "Adaptive mutual exclusion with local spinning," in *DISC2000*, Oct. 2000.

[10] H. Attiya and V. Bortnikov, "Adaptive and efficient mutual exclusion," in *PODC00*, pp. 91–100, 2000.

[11] H. Attiya and A. Fouren, "Adaptive wait-free algorithms for lattice agreement and renaming," in *PODC98*, pp. 277–286, 1998.

[12] H. Attiya and A. Fouren, "Adaptive long-lived renaming with read and write operations," Tech. Rep. 956, Faculty of Computer Science, The Technion, Mar. 1999. www.cs.technion.ac.il/~hagit/pubs/tr0956.ps.gz.

[13] H. Attiya, A. Fouren, and E. Gafni, "An adaptive collect algorithm with applications," in *Distributed Computing*, To appear.

[14] V. Bortnikov, "Adaptive mutual exclusion," Master's thesis, Department of Computer Science, The Technion, Dec. 2000.

[15] M. Choy and A. K. Singh, "Adaptive solutions to the mutual exclusion problem," *DC*, vol. 8, no. 1, pp. 1–17, 1994.

[16] A. Fouren, *Adaptive Wait-Free Algorithms for Asynchronous Shared-Memory Systems*. PhD thesis, Department of Computer Science, The Technion, June 2001.

[17] M. Inoue, S. Umetani, T. Masuzawa, and H. Fujiwara, "Adaptive long-lived $O(k^2)$-renaming with $O(k^2)$ steps," in *DISC2001*, 2001. to appear.

[18] E. Gafni, M. Merrit, and G. Taubenfeld, "The concurrency hierarchy, and algorithms for unbounded concurrency," in *In Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC 2001)*, August 2001.

[19] G. L. Peterson, "Time efficient adaptive mutual exclusion algorithms." Unpublished manuscript, 2000.

[20] M. Moir and J. H. Anderson, "Fast, long-lived renaming," in *WDAG94*, vol. 857 of *LNCS*, pp. 141–155, SV, 1994.

[21] G. Chockler and D. Malkhi, "Active disk paxos," in *To appear in PODC 2002*, 2002.

[22] M. Herlihy and N. Shavit, "The topological structure of asynchronous computability," *Journal of the ACM*, vol. 46(6), pp. 858–923, 1999.

[23] E. Borowsky and E. Gafni, "A simple algorithmically reason characterization of wait-free computations," in *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing*, pp. 189–198, 1997.

[24] G. Hoest and N. Shavit, "Towards a topological characterization of asynchronous complexity," *PODC97*, pp. 199–209, 1997.

[25] 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.

[26] E. Gafni and E. Koutsoupias, "On uniform protocols," *preprint*, 1998.

[27] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merrit, and N. Shavit, "Atomic Snapshots of Shared Memory," in *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, pp. 1–13, 1990.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] E. Borowsky, "Capturing the Power of Resiliency and Set Consensus in Distributed Systems," tech. rep., University of California, Los Angeles, Oct 15, 1995.