LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-475

MORE CHOICES ALLOW MORE FAULTS: SET CONSENSUS PROBLEMS IN TOTALLY ASYNCHRONOUS SYSTEMS

Soma Chaudhuri

October 1992

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

This blank page was inserted to preserve pagination.

More *Choices* Allow More *Faults*: Set Consensus Problems in Totally Asynchronous Systems †

SOMA CHAUDHURI
Laboratory for Computer Science
MIT

September 15, 1992

Abstract

We define the *k-set consensus problem* as an extension of the consensus problem, where each processor decides on a single value such that the set of decided values in any run is of size at most k. We require the *agreement* condition that all values decided upon are initial values of some processor. We show that the problem has a simple $(k-1)$-resilient protocol in a totally asynchronous system. In an attempt to come up with a matching lower bound on the number of failures, we study the *uncertainty* condition, which requires that there must be some initial configuration from which all possible input values can be decided. We prove using a combinatorial argument that any k-resilient protocol for the k-set agreement problem would satisfy the uncertainty condition, while this is not true for any $(k-1)$-resilient protocol. This result seems to strengthen the conjecture that there is no k-resilient protocol for this problem. We prove this result for a restricted class of protocols. Our motivation for studying this problem is to test whether the number of *choices* allowed to the processors is related to the number of *faults*. We hope that this will provide intuition towards achieving better bounds for more practical problems that arise in distributed computing, *e.g.*, the renaming problem. The larger goal is to characterize the boundary between possibility and impossibility in asynchronous systems given multiple faults.

Key words: consensus, impossibility results, fault-tolerance, asynchronous distributed systems

†To appear in *Information and Computation*. A preliminary version appeared in *ACM Principles of Distributed Computing*, August 1990. Supported in part by NSF grants CCR-87-14782 and CCR-89-15206, in part by DARPA contracts N00014-89-J-1988 and N00014-92-J-4033, and in part by ONR contract N00014-91-J-1046.

1 Introduction

The CONSENSUS PROBLEM that has been studied in the literature involves a set of n processors, each with an initial bit (0 or 1), where all these processors have to decide on a common value among themselves. If the initial values of all the processors are 0, they have to decide on 0, while if all the initial values are 1, they have to decide on 1. Additionally, this has to be done in the presence of a number of faults. In a surprising result, Fischer, Lynch and Paterson [FLP83] showed that in a *totally asynchronous* system, the CONSENSUS PROBLEM could not be solved deterministically even in the presence of *one* fault. This is true even though the type of failure considered is the benign form of *fail-stop* fault, where a processor fails by suddenly stopping to function.

Spurred by this negative result, further research focussed on either weakening the model by restricting the asynchronicity ([ADG84], [DDS83], [DLS84]) or strengthening the protocols by allowing randomization ([BO83], [Bra85], [Rab83]). In both cases, consensus protocols were found. But the [FLP83] negative result still seemed to imply that no *decision* could be reached by a *deterministic* protocol in a *totally asynchronous* system.

However, a more recent result [ABND+87] shows that the RENAMING PROBLEM, where each non-faulty processor is making an irrevocable decision, *can* be solved in the presence of *multiple* faulty processors. This would seem very surprising until one realizes that while the CONSENSUS PROBLEM requires decision *and* termination by all the processors, in the RENAMING PROBLEM, the processors are allowed to continue being active *forever* even after they have reached decision. This additional power is useful in solving the RENAMING PROBLEM while it is of no use in solving the CONSENSUS PROBLEM. This leads to the question of identifying specific properties of problems that make them unsolvable in a totally asynchronous system which is subject to faults, to further understand the power of these systems. Other research that has addressed this question is described in [BW87], [BMZ88] and [TKM89]. The first paper studies resource allocation problems and shows that some variants of these problems *are* solvable in the presence of failures. The second paper arrives at a combinatorial characterization of problems that can be solved given a single faulty processor. The third paper studies problems that are solvable in the face of multiple failures. It shows that restricting the set of possible inputs to a problem appropriately would make it solvable in the presence of a larger number of faults.

The [FLP83] proof of impossibility was based on the importance of a "critical event", which, once past, left only one choice to the undecided processors. In the CONSENSUS PROBLEM, this happened when some processor had reached a decision state. In the RENAMING PROBLEM of

[ABND+87], this happens when there are exactly n names and $n-1$ processors have decided on names. Both scenarios seem to imply that in the presence of *one* fault, it is important to have *two* choices available to the undecided processors. Can this idea be extended to multiple faults? In the RENAMING PROBLEM, *it seems* that the size of the name space from which the names are chosen needs to increase with the number of allowable faults, though this is yet to be proven.

The k-SET CONSENSUS PROBLEM that is introduced in this paper is an attempt at testing the idea that the number of *choices* allowed to the processors is directly related to the number of *allowable faults*. It is the natural extension of the CONSENSUS PROBLEM, where, instead of all processors deciding on the same value, they have to decide on at most k different values and terminate. We state the problem below. We have n processors $p_1, p_2, \ldots, p_n$ and each processor p_i has an initial value x_i from a set $V = \{0, 1, \ldots, m-1\}$. Each processor p_i will have to decide on a single value y_i from V such that the collective set of decided values $Y = \{y_i | i \leq n\}$ will be of size *at most* k.

We will need some kind of *non-triviality* condition to make the problem interesting. The non-triviality condition for the consensus problem was that if the initial values of all the processors are the same, that value has to be decided upon by all the processors. Actually, a weaker condition is sufficient for the impossibility proof of [FLP83], that some run must decide 0 and some run must decide 1. This eliminates the possibility that any default value is decided upon. The specific non-triviality condition we will choose here is the *agreement* condition that a value decided upon in any run must be an initial value of some processor. More formally, we require that $Y \subseteq X$ where $X = \{x_i | i \leq n\}$. We will call the variant of the k-set consensus problem including this *agreement* condition the k-SET AGREEMENT PROBLEM. This seems like a natural, and perhaps more useful variant of the set-consensus problem to solve in practice.

We obtain a simple $(k-1)$-resilient protocol for the k-set agreement problem. We conjecture the impossibility of k-resilient protocols for this problem. This result would give credence to our intuition that the number of *choices* allowed in a problem is related to the number of *faults*. We have proved this result for a restricted class of protocols, called *stable-vector protocols*. For the unrestricted case, we have only shown the relatively trivial lower bound of the impossibility of a $nk/(k+1)$-resilient protocol for k-set agreement.

In an attempt at proving the general impossibility result, we introduce an additional *uncertainty* condition on any protocol solving the k-set-consensus problem, which says that for some vector $(x_1, x_2, \ldots, x_n)$ of initial values of the processors, there are at least $k+1$ different values in V such

that each is decided upon in some run of that protocol starting at the initial values specified by the vector. That is, the set of values decided upon is not always pre-determined given the set of initial values of the processors, but is determined in the course of the run. This uncertainty condition exists in the canonical consensus problem, and was important in obtaining the impossibility result in [FLP83]. We introduce it here to see if it exists in the k-set agreement problem, and to see if it helps us in obtaining our impossibilty result.

We prove, by a non-trivial combinatorial argument involving Sperner's Lemma [Spe28], that any k-resilient protocol for k-set agreement must satisfy the uncertainty condition. This is however not true for any t-resilient protocol, where $t < k$. This is the main result of the paper, and we believe this is a first step towards proving our key conjecture that no k-resilient protocol exists for the problem in a totally asynchronous system.

While the [FLP83] result is based on the *asynchronous message-passing* model, Loui and Abu-Amara [LAA87] showed the same impossibility result in the *shared memory* model. Our result does not assume either the message-passing or the shared-memory paradigm, and will hold in either case. It is a general result based only on the possibility of fail-stop faults. In fact, it does not even assume a totally asynchronous system, and holds for both synchronous and asynchronous systems.

Our algorithms, on the other hand, will assume a totally asynchronous system, and either the message-passing or the shared-memory model for communication.

2 Preliminaries

We will first define our problem and describe the model. The k-set consensus problem is as follows: In an asynchronous system of n processors $p_1, p_2, \ldots, p_n$, each processor p_i starts with an input value $x_i \in V$. It follows a deterministic transition function by which it communicates with other processors, and eventually *decides* on an output value $y_i \in V$. We define the set of possible input/output values $V = \{0, 1, \ldots, m-1\}$, where $m > k$. Each processor p_i is modeled as an infinite-state machine with state set $\mathcal{Z}$. There are m possible *initial states*, $I_0, I_1, \ldots I_{m-1} \in \mathcal{Z}$, denoting the initial values. Processor p_i starts in initial state I_{x_i}. There are m sets of *decision* states $\mathcal{F}_0, \ldots, \mathcal{F}_{m-1} \subset \mathcal{Z}$. Each processor p_i eventually enters a decision state in $\mathcal{F}_{y_i}$, effectively *deciding* on the value y_i. Once a processor enters a state in $\mathcal{F}_v$, for some v, further transitions can only lead to other states in $\mathcal{F}_v$. This models the requirement that a decision once reached by a processor is irrevocable. Moreover, modelling a set of decision states (rather than one state)

for each decision value allows the processors to perform non-trivial actions even after reaching a decision.

There are two fundamental models for processor communication, *shared memory* and *message passing*. Our impossibility result is valid in both models, and we describe them below. In the message passing model, processors communicate by receiving and sending messages. Each processor has a *buffer* associated with it, which contains all the messages that have been sent to it but have not yet been received.

There are two possible operations supported by the buffers:

1. **Send** (p, m) places message m in processor p's buffer.
2. **Receive** (p) deletes some subset of messages μ from p's buffer and delivers it to p. The subset of messages delivered may be empty even if the buffer is non-empty.

In the shared memory model, processors communicate by reading and writing into shared memory registers. Given registers $X_1, \ldots, X_l$, the operations are as follows:

1. $\mathbf{Read}_p(X_i)$ is the read operation by processor p from register X_i, returning the value of the register.
2. $\mathbf{Write}_p(X_i, v)$ is the write operation by processor p that writes value v into register X_i.

A *configuration* of the system consists of the state of each processor. In a *message passing* system it also includes the contents of each message buffer. In a *shared memory* system it also includes the values of each shared register. The *initial configuration* consists of the initial states of each processor (I_{x_i}, for all i), the empty message buffers in the case of *message passing*, and the initial values of the registers in the case of *shared memory*.

A *step* is defined to be the primitive step of a single processor p, changing the configuration of the system. In a *message passing* system, the step occurs in two phases. First, *receive* (p) is performed, which causes a (possibly empty) set of messages μ to be delivered to p. Based on the messages received, processor p changes state and executes its protocol, sending messages to the other processors using the operation *Send* (q, m). In a *shared memory* system, either a *read* or a *write* operation by a processor, followed by a change in state, constitutes a step.

Our asynchronous model can be described as follows. Since we allow the processors to be completely asynchronous, each processor can wait an arbitrary amount of time before performing

a step. However, every reliable processor will perform a step an infinite number of times or until it terminates. Also, since communication is reliable, though asynchronous, in the *message passing* model, a given message placed in processor p's buffer will be eventually delivered given that *receive* (p) is performed an infinite number of times, though it may not be delivered after any given finite number of *receive* (p) executions. In other words, any reliable processor will eventually receive all the messages sent to it. It is important to note that *message order* is not preserved; a message that was placed in the message buffer earlier could be delivered later. In the *shared memory* model, no additional conditions are placed on the system.

A few more definitions are necessary to formalize the model. Note that while each processor behaves deterministically, the non-determinism or *uncertainty* is introduced by the relative order of steps among the processors, and the message delivery system in the case of *message passing.*

Definition 2.1 *A configuration D is* **reachable in one step** *from configuration C, if a step taken by some processor in configuration C results in configuration D.*

Definition 2.2 *A configuration C_k is* **reachable** *from C_0 if there is a finite sequence $C_0, \ldots, C_k$ such that for all $0 \leq i < k$, C_{i+1} is reachable in one step from C_i.*

A *run* describes the sequence of configurations that the system goes through as it executes the protocol:

Definition 2.3 *A* **run** *is a sequence (finite or infinite) of configurations $C_0 C_1 \ldots C_k \ldots$ where C_0 is an initial configuration and for all $i \geq 0$, C_{i+1} is reachable in one step from C_i.*

We say that a processor is *non-faulty in an infinite run* if it performs a step infinitely many times in that run. Otherwise, it is *faulty.* Note that the type of faults we consider here are *fail-stop faults.* A processor, if it takes a step, will behave as specified by its protocol.

A run of a k-resilient protocol can have at most k faults. We *label* each configuration by the possible values that can be decided (by any processor) in any configuration reachable from it. We define the k-valent property of a configuration along the same lines as the univalent and bivalent properties of [FLP83].

Definition 2.4 *The set $L \subseteq V$ is the* **label** *of a configuration C, i.e. $L =$* **label**(C)*, where*

$L = \{v \in V |$ *there is a configuration reachable from* C *where some processor decides* $v\}$

Definition 2.5 *A configuration* C *is said to be* k**-valent** *if* $|\mathsf{label}(C)| \geq k$.

Note that for non-triviality we required that $|V| \geq k+1$. Any protocol for the k-set consensus problem satisfies the following *validity* condition:

k-validity: The set of values Y decided upon in any run is such that $Y \subseteq V$ and $|Y| \leq k$.

A further non-triviality condition that we introduce is the *agreement* condition, which requires that any value decided in a run must be an input value of some processor in that run. We discuss in Section 5 why this particular condition, apart from being a useful and natural condition, is essential for our purposes. We define the k-SET AGREEMENT PROBLEM to be the k-set consensus problem with the agreement condition, and this is the problem we will concentrate on.

agreement: If the set of initial values in a given run is $X \subseteq V$, the set of decided values is $Y \subseteq X$.

The k-set agreement problem seems like a good candidate to test our intuition that the number of choices allowed to the processors in a problem is related to the number of faults that can be tolerated by a protocol for the problem. The k-set agreement problem allows k choices. If our intuition were to be correct, there should be a $(k-1)$-resilient protocol for the problem, but no k-resilient protocol should exist.

In Section 4, we will present a $(k-1)$-resilient protocol for k-set agreement. We will define a restricted class of protocols, called *stable-vector* protocols. Our $(k-1)$-resilient protocol is such a protocol. We will then prove that no k-resilient stable-vector protocol exists for k-set agreement.

While the lower bound result referred to above holds for a restricted class of protocols, we would like to achieve the same lower bound for *all* protocols. Towards proving this general matching lower bound, we introduce the *uncertainty condition*, given below.

uncertainty: There exists a $(k+1)$-valent initial configuration.

In Section 3, we prove that any k-resilient protocol for the k-set agreement problem satisfies the uncertainty condition. The result requires a combinatorial proof involving Sperner's Lemma [Spe28].

Why is this result interesting? Our goal is to prove the impossibility of a k-resilient protocol for k-set agreement. Showing that any k-resilient protocol for k-set agreement must satisfy the uncertainty condition, *i.e.*, it must have a $(k+1)$-valent initial configuration, is one step towards this goal. Recall that the impossibility proof in [FLP83] involved a similar step. It first proved that any 1-resilient protocol for consensus must have a bivalent initial configuration. It then proved inductively that it was possible to remain in a bivalent configuration indefinitely, while taking all the legal steps of the protocol. The hope is that a similar inductive argument might prove useful to show impossibility of the k-set agreement problem in the presence of k faults.

3 Agreement Implies Uncertainty With k Faults

We come to our main technical result, which shows that any k-resilient protocol for k-set agreement must satisfy the uncertainty condition. As we stated earlier, this result holds for both synchronous and asynchronous systems.

Theorem 3.1 (Uncertainty Theorem) *In a distributed system with fail-stop faults, for any k-resilient protocol which solves the k-set agreement problem, there exists a $(k+1)$-valent initial configuration.*

The notion of a *k-orbit* is important in our proof. We define it here.

Definition 3.1 *A* **k-orbit** *is a set of initial configurations which differ in the initial values of at most k processors.*

Lemma 3.2 *For any k-resilient protocol, let $\{C_1, \ldots, C_m\}$ be a k-orbit, and for all $i \leq m$, let L_i be the label of C_i. Then, $L_1 \cap L_2 \cap \cdots \cap L_m$ is non-empty.*

Proof: Let $p_1, \ldots, p_k$ be the processors whose values differ in the initial configurations in the k-orbit. Consider an identical run starting from each of these configurations in which these k processors are initially dead. The remaining processors should all decide in these runs since our protocol is k-resilient. Furthermore, the same set of values L must be decided in all these runs since they are identical. Therefore, L is contained in L_i, for all $i \leq m$, and it follows that $L_1 \cap L_2 \cap \cdots \cap L_m$ must be non-empty. □

We will prove the theorem for the case $k = 2$. This will give a flavor of the general proof, while retaining some simplicity. The proof for $k > 2$ is substantially more complicated, and will be dealt with in a separate subsection. Our proof makes use of Sperner's Lemma [Spe28], which we state below.

Lemma 3.3 (Sperner's Lemma) *Given a triangle with arbitrary points in its interior and boundaries, color each vertex of the triangle and all interior and boundary points with three colors following the coloring restriction given below:*

- *The 3 vertices of the triangle must be colored by the 3 different colors.*
- *Every point on an edge of the triangle must take on the color of one of the two vertices adjacent to that edge.*

Given any triangulation of the triangle with respect to its interior and boundary points, consider the unit triangles it forms. There are an odd number of such unit triangles whose vertices are colored with three distinct colors. In particular, there exists at least one such unit triangle.

Proof of Theorem 3.1 for the case $k = 2$:

Let P be a 2-resilient protocol for the problem. We form a triangle on a grid whose vertices, interior and boundary points represent initial configurations of the protocol. Each initial configuration is defined by an n-tuple of initial values $(x_1, \ldots, x_n)$ where each entry x_i is some value in $V = \{a, b, c\}$. Note that without loss of generality, we can assume $|V| = 3$, since impossibility in this case would imply impossibility in general. (An algorithm that works if $|V| = m$ will clearly work if $|V| < m$.) The case for $n = 5$ is illustrated in Figure 1.

Not all the initial configurations are represented in this triangle. The configuration $(x_1, \ldots, x_n)$ is represented if there exists some i, j, where $0 \leq i \leq j \leq n$, such that $x_l = a$ for $l \leq i$, $x_l = b$ for $i < l \leq j$, and $x_l = c$ for $l > j$. The point (i, j) in our grid is occupied by that initial configuration. Note that the initial configuration $(b, \ldots, b)$ occupies the $(0, n)$ position, $(c, \ldots, c)$ occupies the $(0, 0)$ position and $(a, \ldots, a)$ occupies the (n, n) position. These are the three vertices of the triangle and all points (i, j) where $0 \leq i \leq j \leq n$ are interior or boundary points of the triangle. Intuitively, every move up in the y-axis corresponds to a change of one entry from c to b, and every move to the right on the x-axis corresponds to a change of one entry from b to a.

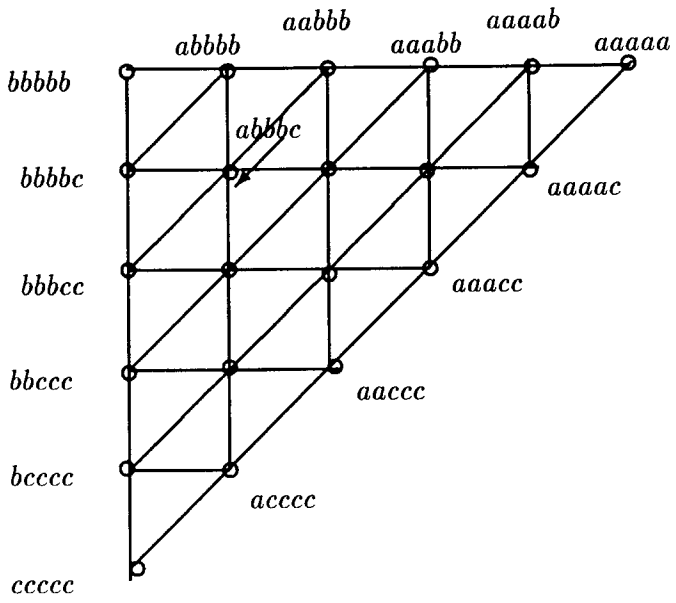

Figure 1: A Triangle Illustrating the Application of Sperner's Lemma

Consider the simple triangulation of this triangle formed by bisecting each unit square, as shown in Figure 1. We claim that each unit triangle defines a 2-orbit. This is obvious, since any three initial configurations C_1, C_2, C_3 which form a unit triangle clearly differ in the values of exactly two processors. Let the labels of these configurations be L_1, L_2, L_3. By Lemma 3.2, we know that $L_1 \cap L_2 \cap L_3$ is non-empty.

Now, to continue with the proof of the theorem, suppose there is no 3-valent initial configuration. Then, each configuration represented in the triangle has a label from the set $\{a, b, c, ab, ac, bc\}$. We color the triangle with the three colors K_0, K_1 and K_2 in the following manner. Any configuration with label c or bc is colored K_0, those with label a or ac are colored K_1, and those with labels b or ab are colored K_2. We need to check if our coloring satisfies the conditions of Sperner's Lemma. Since the three vertices of the triangle must have labels a, b and c by the agreement condition, they are colored with the three different colors K_1, K_2 and K_0. Any point on the edge joining the vertices colored K_1 and K_2 must have the label a, b or ab, again by the agreement condition. Therefore, such a point would be colored either K_1 or K_2. The same holds for the other edges. Now, applying Sperner's Lemma, we have the result that there exists some unit triangle which is colored by K_1, K_2 and K_0. Then, the labels of the three corresponding configurations have an empty intersection.

This leads to a contradiction since we claimed earlier that the intersection of the labels must be non-empty. □

3.1 The Generalized Proof of the Uncertainty Theorem

To prove the result in general for all values of k, we need the general form of Sperner's Lemma [Spe28]. The one listed above is for the case where $k = 2$. The general form argues about k-simplices rather than triangles (which are 2-simplices) and $(k + 1)$ colors instead of three. While for the $k = 2$ case, the lemma was directly applicable to obtain our result, this is not true for larger values of k. The intuitive reason for this added complexity is that while for $k = 2$, a simple triangulation gave rise to unit triangles which could be shown to be 2-orbits, it is harder to argue for $k > 2$ that a simplicial decomposition exists which produces unit k-simplices which correspond to k-orbits.

We first describe some definitions relating to simplices. A **k-simplex** is a convex hull of $k + 1$ affinely independent points, which are called its **vertices**. Note that a 0-simplex is a point, a 1-simplex is a line segment, a 2-simplex is a triangle and a 3-simplex is a tetrahedron.

For all $l < k$, an l-simplex is a **face** of a k-simplex if all the vertices of the l-simplex are vertices of the k-simplex. This is consistent with the geometric notion that the faces of a tetrahedron are triangles.

We now proceed to give the general proof of the theorem. First we state the generalized form of Sperner's Lemma [Spe28].

Lemma 3.4 (Generalized form of Sperner's Lemma) *Given a k-simplex with arbitrary points in its interior and boundaries, color each vertex of the k-simplex and all interior and boundary points with $k + 1$ colors following the coloring restriction given below:*

- *The $k + 1$ vertices of the k-simplex must be colored by the $k + 1$ different colors.*
- *If a point is contained in an l-simplex which is a face of the k-simplex, where $l \leq k$, it must take on the color of one of the $l + 1$ vertices of the l-simplex.*

Given any simplicial decomposition of the k-simplex with respect to its interior and boundary points, consider the unit k-simplices it forms. There are an odd number of such unit simplices whose $k + 1$ vertices are colored with $k + 1$ distinct colors. In particular, there exists at least one such unit k-simplex.

Generalized Proof of the Uncertainty Theorem:

Let P be a k-resilient protocol for the problem. We will prove, by contradiction, that a $(k+1)$-valent initial configuration must exist.

We form a k-simplex in the Cartesian Coordinate System whose vertices, interior and boundary points represent initial configurations of the protocol. Without loss of generality, we can assume that $|V| = k+1$, since an impossibility result in this case would imply impossibility whenever $|V| > k$. So, we define the set of initial values of the processors to be $V = \{0, \ldots, k\}$. Each initial configuration is defined by an n-tuple of initial values $(x_1, \ldots, x_n)$ where each entry x_i is in V.

Not all the initial configurations are represented in the k-simplex. The set of initial configurations I_{mon} represented have the following property. For all i, j such that $0 \leq i \leq j \leq n$, $x_i \leq x_j$. In other words, there is a *monotonic* order in the initial values of the processors.

We define a one-to-one mapping f between the initial configurations and the points in the k-dimensional Cartesian grid. For every initial configuration $x = (x_1, \ldots, x_n) \in I_{mon}$, $f(x) = z = (z_1, \ldots, z_k)$, where for all $i \leq k$, $z_i = j$, such that $x_j < i$ and $x_{j+1} \geq i$. In other words, z_i is the number of processors whose values in the initial configuration x are less than i.

These points define a k-simplex. Note that the point $(0, \ldots, 0)$ represents the configuration $(k, \ldots, k)$. Each configuration of the form $(l, \ldots, l)$, where $l < k$, is represented by the point $(z_1, \ldots, z_k)$, where for all $i \leq l$, $z_i = 0$, and for all $i > l$, $z_i = n$. These $k+1$ points are the vertices of the k-simplex. Each dimension of the k-simplex is of length n. The interior and boundary points correspond to the points on the grid. Let $\mathcal{Z}$ be the set of integers. Note that the k-simplex contains all points $(z_1, \ldots, z_k)$ such that for all i, $z_i \in \mathcal{Z}$, and for all i, j where $i \leq j \leq k$, $0 \leq z_i \leq z_j \leq n$.

We define the following triangulation [Tod76] over $\mathcal{Z}^k = \{(z_1, \ldots, z_k) | z_i \in \mathcal{Z}\}$, the k-dimensional space, where $\mathcal{Z}$ is the set of integers. This is known as the Kuhn triangulation. For all $j \leq k$, let $u^j = (u_1, \ldots, u_k)$ such that $u_j = 1$ and for all $i \neq j$, $u_i = 0$. In other words, u^j is the jth unit vector of length k. Let π be any permutation function of k, *i.e.*, a one-to-one function from $\{1, \ldots, k\}$ to itself. For each point, $y^0 \in \mathcal{Z}^k$, and permutation π, we define the unit k-simplex with vertices $y^0, y^1, \ldots, y^k$, where, for all i, $y^i = y^{i-1} + u^{\pi(i)}$. The collection of all such unit k-simplices defined over all points $y^0 \in \mathcal{Z}^k$ and all permutation functions π of k, corresponds to our triangulation.

The following lemma states that Kuhn's simplicial decomposition of the k-simplex creates unit k-simplices, each of which corresponds to a k-orbit. The lemma will be proven later.

Lemma 3.5 *Kuhn's triangulation divides the k-simplex into unit k-simplices such that each unit k-simplex corresponds to a set of $k+1$ initial configurations which defines a k-orbit.*

Now, suppose, for contradiction, that there exists no initial $(k+1)$-valent configuration. Then for each initial configuration C in I_{mon}, $|\mathsf{label}(C)| \leq k$, where $\mathsf{label}(C) \subseteq V$ and $V = \{0, \ldots, k\}$.

Consider the set of $k+1$ colors $\mathcal{K} = \{K_0, \ldots, K_k\}$. The color function maps each initial configuration in I_{mon}, *i.e.*, each point in the k-simplex, to a color in $\mathcal{K}$. We define it below. First, for all $i \in V$, define $\mathsf{next}(i) = (i+1) \bmod (k+1)$. Now, let $\mathsf{next\text{-}label}(C) = \{\mathsf{next}(i) | i \in \mathsf{label}(C)\}$. Also, $\overline{\mathsf{label}(C)} = V - \mathsf{label}(C)$. Then $\mathsf{color}(C) = K_{index}$ where

$$index = \mathsf{min}(\overline{\mathsf{label}(C)} \cap \mathsf{next\text{-}label}(C)).$$

Note that $\overline{\mathsf{label}(C)} \cap \mathsf{next\text{-}label}(C)$ is always non-empty. This is because, since $\mathsf{label}(C) \neq V$, there exists some value $i \in \mathsf{label}(C)$ such that $\mathsf{next}(i) \notin \mathsf{label}(C)$. Also, *index* can be defined to be any value in the set; we arbitrarily define it to be the minimum value.

Note that our proof for the special case $k = 2$ follows this coloring scheme with the value set $V = \{a, b, c\}$ renamed to $\{0, 1, 2\}$, where a corresponds to 0, b to 1, and c to 2. We state the following lemmas related to the color mapping, which will be proven later.

Lemma 3.6 *The mapping* color *is a coloring of the k-simplex which satisfies the conditions of Sperner's Lemma.*

Lemma 3.7 *If a unit k-simplex contains all the colors in $\mathcal{K}$, then the intersection of the labels of the corresponding set of $k+1$ initial configurations is empty.*

By Lemma 3.6, the coloring satisfies the conditions of Sperner's Lemma. By Lemma 3.7, if a unit k-simplex contains all the colors in $\mathcal{K}$, then the intersection of the labels of the corresponding set of $k+1$ initial configurations is empty. But, this set of initial configurations corresponds to an orbit by Lemma 3.5, and by Lemma 3.2, the intersection of the labels is non-empty. Therefore, no unit k-simplex can have all the colors in $\mathcal{K}$. Since the coloring satisfies the conditions of Sperner's Lemma, this contradicts the lemma, which states that at least one unit k-simplex must contain all the $k+1$ colors. Therefore, there must exist an initial $(k+1)$-valent configuration. □

We now prove the lemmas required in proving the theorem.

Proof of Lemma 3.5:
Note that Kuhn's triangulation divides each unit k-hypercube independently into $k!$ unit k-simplices. Each unit k-simplex is part of exactly one unit k-hypercube. We label each k-hypercube by its

lowest point. The unit hypercube defined by the vertices $z = (z_1, \ldots, z_k)$, for all z such that for all i, $z_i \in \{y_i, y_i + 1\}$, has the label $(y_1, \ldots, y_k)$, which is its lowest point. The hypercube labeled y^0 is divided into $k!$ unit simplices defined by y^0 and each of the $k!$ permutations of k.

We call the set of unit k-simplices defined by Kuhn's triangulation SIM. Kuhn [Kuh60] shows that the set SIM of unit k-simplices is indeed a simplicial decomposition of $\mathcal{Z}^k$. Since our k-simplex is a convex subset of $\mathcal{Z}^k$, we prove that a subset of the set SIM of unit k-simplices form a triangulation of our k-simplex X. Clearly, any two unit simplices are mutually exclusive, *i.e.* they do not share any internal points, but only boundary points. To prove that there exists a subset of unit simplices such that the union of the unit simplices equals our k-simplex X, it is sufficient to prove that every unit simplex in SIM is either completely contained in X or is completely outside X. In other words, for every point $r \in \mathcal{R}^k$ which is contained in X, if r is not contained in any face of a unit k-simplex, then all the vertices of the unit simplex containing it must be in X. We prove this in the following.

Claim 3.8 *If a point $r \in \mathcal{R}^k$ is contained in X, but not contained in any face of a unit k-simplex in SIM, then all the vertices of the unit k-simplex containing it must be in X.*

Proof: Let $r = (r_1, r_2, \ldots, r_k) \in \mathcal{R}^k$ be a point in X. Note that the fact that r is not contained in any face of a unit k-simplex implies that for all i, $r_i \notin \mathcal{Z}$. Therefore, for all i, $\lfloor r_i \rfloor < r_i < \lceil r_i \rceil$.

The point r belongs to the unit k-hypercube H labeled $y^0 = (\lfloor r_1 \rfloor, \lfloor r_2 \rfloor, \ldots, \lfloor r_k \rfloor)$. Let *sim* be the *unique* unit simplex containing r, defined by the vertices $y^0, \ldots, y^k$. For all $l \leq k$, define for all i, $[y^l]_i \in \mathcal{R}$ such that $y^l = ([y^l]_0, \ldots, [y^l]_k)$. Note that, by the definition of the decomposition, for all l, l', i such that $l \leq l' \leq k$, $[y^l]_i \leq [y^{l'}]_i$.

If H is completely contained in X, clearly all the vertices of *sim* must be in X. Suppose H is not completely contained in X. We suppose, for contradiction, that there exists some $i \leq k$ such that y^i is not contained in X. Let $b \leq k$ be the smallest integer such that y^b is not contained in X.

Suppose that $b = 0$. Then, y^0 is not in X. Recall that X contains all points $(z_1, \ldots, z_k)$ such that for all i, $z_i \in \mathcal{Z}$, and for all i, j where $i \leq j \leq k$, $0 \leq z_i \leq z_j \leq n$. This implies that for some $i \leq k$, $[y^0]_i > [y^0]_{i+1}$. But, this would imply that $r_i > r_{i+1}$, which is a contradiction of the fact that r is in X.

Therefore, $b > 0$. Recall that for all $l, m \leq k$, $[y^l]_m \in \{\lfloor r_m \rfloor, \lceil r_m \rceil\}$, and, $\lfloor r_m \rfloor + 1 = \lceil r_m \rceil$. So, for all i, the ith bit takes on one of two consecutive values in each of the vectors y^l.

For all l, m such that $l \leq m$, $r_l \leq r_m$. Also, for all l, $r_l < n$. This follows from the definition of our k-simplex X and of r. Therefore, it follows that for all l, m, $[y^l]_m \leq n$, since $r_m < [y^l]_m$, for all l, m.

Let $\pi(b) = a$, where π is the permutation used in computing the vertices $y^0, \ldots, y^k$ of the unit simplex sim. Then, $y^b = y^{b-1} + u^a$. Note that $[y^{b-1}]_a + 1 = [y^b]_a$, and for all $m \neq a$, $[y^{b-1}]_m = [y^b]_m$. So, the two vectors differ at only the ath bit.

Since y^{b-1} is in X, it follows that for all l, m such that $l \leq m$, $[y^{b-1}]_l \leq [y^{b-1}]_m$. Now, since y^b is not in X, and y_b and y_{b-1} differ in only the ath bit, and we know that $[y^b]_a \leq n$, it follows that $[y^b]_a > [y^b]_{a+1}$.

Now, by the relationship of y^b and r, we have $\lfloor r_a \rfloor + 1 = [y^b]_a$. Therefore, $\lfloor r_a \rfloor + 1 > [y^b]_{a+1}$, and $\lfloor r_a \rfloor \geq [y^b]_{a+1}$. We know that $\lfloor r_a \rfloor \leq \lfloor r_{a+1} \rfloor$. Therefore, $\lfloor r_{a+1} \rfloor \geq [y^b]_{a+1}$. However, $\lfloor r_{a+1} \rfloor \leq [y^b]_{a+1}$. Therefore, it follows that $\lfloor r_{a+1} \rfloor = [y^b]_{a+1}$. So, we have that $\lfloor r_a \rfloor + 1 > \lfloor r_{a+1} \rfloor$. Since $\lfloor r_a \rfloor \leq \lfloor r_{a+1} \rfloor$, it follows that $\lfloor r_a \rfloor = \lfloor r_{a+1} \rfloor$.

It follows that for all $i < b$, $[y^i]_a = [y^i]_{a+1}$, and for all $i > b$, $[y^i]_a \geq [y^i]_{a+1}$.

Now, by a fact of linear algebra, we know that r must be a positive linear (convex) combination of $y^0, \ldots, y^k$. In other words there exist positive constants $c_0, \ldots, c_k$, such that

$$r = c_0 \circ y^0 + \cdots + c_k \circ y^k$$

Given a defined above, we have

- $r_a = c_0 \circ [y^0]_a + \cdots + c_k \circ [y^k]_a$
- $r_{a+1} = c_0 \circ [y^0]_{a+1} + \cdots + c_k \circ [y^k]_{a+1}$

We showed above that $[y^b]_a > [y^b]_{a+1}$, and for all $i \neq b$, $[y^i]_a \geq [y^i]_{a+1}$. Therefore, $r_a > r_{a+1}$, which is a contradiction. □

The result of the lemma follows from the claim. □

We now prove two properties about the coloring, stated in Lemmas 3.6 and 3.7.

Proof of Lemma 3.6:
The two conditions of Sperner's Lemma are as follows. First, the $k+1$ vertices of the k-simplex must be colored with $k+1$ distinct colors. We prove this fact. The labels of the $k+1$ vertices are $\{0\}, \{1\}, \ldots, \{k\}$. By the definition of **next** and **color**, their corresponding colors in $\mathcal{K}$ will therefore be $K_1, K_2, \ldots, K_k, K_0$, respectively. Therefore, the first condition is satisfied.

The second condition of Sperner's Lemma states that if a point in the k-simplex is contained in an l-simplex which is a face of the k-simplex, then its color must be the color of one of the vertices of the l-simplex. Let C be a configuration that lies in an l-simplex, where $l < k$. Let $\textsf{color}(C) = K_j$, where $j \in V$. By the definition of **color**, it follows that $j \in \textsf{next-label}(C)$. Let $i \in V$ be such that $\textsf{next}(i) = j$. By the definition of **next-label**, this implies that $i \in \textsf{label}(C)$. Now, by the agreement condition, there exists a configuration D which corresponds to a vertex of the l-simplex such that $\textsf{label}(D) = \{i\}$. Therefore, $\textsf{next-label}(D) = \{j\}$ and $\textsf{color}(D) = K_j$. So, D is a vertex of the l-simplex with the same color as C, and we have our result. □

Proof of Lemma 3.7:
Suppose there is a unit k-simplex containing all the colors in $\mathcal{K}$. Let the set of configurations that defines the unit k-simplex be $\{C_0, C_1, \ldots, C_k\}$, where, for all i, C_i has the color K_i.

We define a one-to-one correspondence between each color $K_i \in \mathcal{K}$ and a label of size k. For all i, K_i corresponds to the label $L_i = V - \{i\}$.

Now, for all i, $\textsf{label}(C_i)$ is the label of C_i. We claim that the intersection of all the $\textsf{label}(C_i)$'s is empty. We first prove that for all i, $\textsf{label}(C_i) \subseteq L_i$.

Note that by definition of **color**, for all i, the fact that C_i has color K_i implies that $i \notin \textsf{label}(C_i)$. Since i is the only element of V which is not contained in L_i, it follows that $\textsf{label}(C_i) \subseteq L_i$.

Clearly, $\bigcap_i L_i = \emptyset$. Since for all i, $\textsf{label}(C_i) \subseteq L_i$, it follows that $\bigcap_i \textsf{label}(C_i) = \emptyset$. □

This completes our proof of the Uncertainty Theorem.

3.2 Other Impossibility Results

We define the EXACTLY-k-SET CONSENSUS PROBLEM to be the problem where a set of n processors start with an input value within the set V, where $V > k$. Each processor p_i decides on a value

y_i, such that the set of decided values $Y = \{y_1, \ldots, y_n\}$ is of cardinality *exactly* k. In addition, all values must be decided in some run. We claim that the exactly-k-set consensus problem does not even have a 1-resilient protocol. We conclude this section by giving an obvious lower bound for the k-set agreement problem in the asynchronous *message-passing* model.

Theorem 3.9 *There is no* 1*-resilient protocol for the exactly-*k*-set consensus problem.*

The proof follows along similar lines to [FLP83], and is omitted here for brevity.

Theorem 3.10 *There is no* $\lceil nk/(k+1) \rceil$*-resilient protocol for* k*-set agreement in the message passing model.*

Proof: Suppose there exists such a protocol. Divide the set of processors into $k+1$ groups $G_0, \ldots G_k$, so that each group has either $\lceil n/(k+1) \rceil$ processors or $\lfloor n/(k+1) \rfloor$ processors. Now, consider an initial configuration in which all the processors in group G_i have input value i, for all $i \leq k$. Given a run where each processor only hears from (either directly or indirectly) other processors which share its initial value, every processor must decide on its own value. Then, $(k+1)$ different values are decided upon, which violates k-set agreement. □

4 Stable Vector Protocols for k-set Agreement

We present a simple $(k-1)$-resilient protocol which solves the k-set agreement problem, in a totally asynchronous system with fail-stop faults. It has some similarity to the algorithm for the renaming problem presented in [ABND+87]. We present this protocol in the message passing model. We will later present a few protocols in the shared memory model.

Theorem 4.1 *There exists a* $(k-1)$*-resilient protocol for the* k*-set agreement problem where* $n > 2(k-1)$ *and* n *is the total number of processors.*

The algorithm is as follows. Each processor maintains a vector of size n where the ith entry corresponds to the initial value of processor i, if it knows it, and the value ϕ if it doesn't. Initially, the vector of each processor will only contain its own initial value. In each round, each processor broadcasts its vector and then receives the vectors of other processors and incorporates the new information into its own vector. Note that since $k-1$ processors may be faulty, a processor can

For processor p_i:

1. Construct initial vector T_i such that $T_i(i) = x_i$ and $\forall j \neq i$, $T_i(j) = \phi$.
2. Broadcast T_i to all processors and set $r := 1$.
3. If a vector T_j for some j *which is not a decider vector* is received,
   (a) If $T_j \prec T_i$, goto 3.
   (b) If $T_j = T_i$, then
      i. set $r := r + 1$
      ii. if $r < n - k + 1$ then goto 3, else goto 5.
   (c) If $T_j \not\preceq T_i$, $T_i :=$ **update** (T_i, T_j) and goto 2.
4. Otherwise, if a decider vector T_j is received, set $T_i := T_j$.
5. Decide on y_i where $y_i =$ **compute** (T_i). Broadcast a message containing T_i *labeled as a decider vector* to all processors and terminate.

Figure 2: The k-Set Agreement Algorithm

only wait for $n-k+1$ vectors in each round. When a processor reaches a round in which it receives $n - k + 1$ vectors that are identical to its own vector, it stops and decides on a value based on this vector. It then broadcasts this vector, labeling it as a *decider vector.*

Before we can state the algorithm in more formal terms, some notation has to be clarified. Let $T(i)$ be the ith entry of the vector T. We can define partial orders $\preceq$ and $\prec$ on the vector as follows:

Definition 4.1 *Given vectors* T_1 *and* T_2, $T_1 \preceq T_2$ **if** $\forall i \leq n$, $T_1(i) = \phi \bigvee T_1(i) = T_2(i)$. *Also,* $T_1 \prec T_2$ **if** $T_1 \preceq T_2 \bigwedge T_1 \neq T_2$.

Intuitively, T_2 has all the information of T_1 and perhaps more. We also define the notion of a *complete* vector and a *t-eligible* vector.

Definition 4.2 *A vector* T *is* **complete** *if* $\forall i \leq n$, $T_i \neq \phi$. *A vector* T *is* t**-eligible** *if it has at most* t *entries with value* ϕ.

We describe the Set-Agreement Algorithm for processor p_i (see Figure 2).

We need to define the two functions, **update** and **compute**, used in this algorithm. The function **update** takes as input two consistent vectors T_1 and T_2, each of length n and outputs a new vector T such that $\forall i \leq n$,

1. if $T_1(i) = \phi \bigwedge T_2(i) = \phi$ then $T(i) = \phi$.

2. if $T_1(i) = x_i \bigvee T_2(i) = x_i$ then $T(i) = x_i$.

Note that $T_i, T_j \preceq$ **update** (T_i, T_j).

The function **compute** maps each $(k-1)$-*eligible* vector S into a value in V. We define **compute** (S) to satisfy the agreement condition as follows. Every vector is mapped arbitrarily to any value present in the vector. This is enough to guarantee agreement.

It is notable here that depending on the specific mapping of the vectors, the algorithm may or may not satisfy the uncertainty condition. This shows that the Uncertainty Theorem does not hold for $(k-1)$-resilient protocols.

We need to argue the correctness and termination of this protocol. First, we show termination. The protocol for a processor p_i terminates when, after broadcasting some vector T_i, it either receives $n-k+1$ messages identical to T_i, or it receives a decider vector. Once one processor sends a decider vector, every processor will eventually receive it, unless it has already terminated. So, it is sufficient to show that some processor must decide. For sake of contradiction, we assume that none of the processors terminate.

Lemma 4.2 *Some processor will eventually receive $n-k+1$ messages identical to its own vector in every run.*

Proof: Suppose, in some run, no processor ever receives $n-k+1$ identical messages. Consider processor p_i and let T_i be the maximal vector broadcast by p_i over the course of this run. Note that such a maximal vector must exist. Let p_i broadcast T_i. Since, by assumption, none of the non-faulty processors have terminated, they will all eventually receive T_i. Each processor p_j will then broadcast a vector T_j such that $T_i \preceq T_j$. For all j, p_i will receive T_j. Since T_i was p_i's maximal vector, $T_i \not\prec T_j$. Therefore, for all j, $T_j = T_i$. Since there are at least $n-k+1$ non-faulty processors, we have a contradiction. □

Before we prove correctness, we first define the concept of a *stable* vector associated with a run. For any given run, a vector T is a **stable** vector if some processor p_i has received $n-k+1$ copies of T in that run of the protocol. While there is a partial order defined on the set of all vectors, we will show that the set of stable vectors constitute a total order. The idea of a stable vector was earlier shown in [ABND+87].

Each processor computes its decision value based on the vector it terminates with, and each of these vectors are *stable* vectors. Therefore, it is sufficient to prove that there can be at most k possible stable vectors in any run, since that implies that there are at most k different decisions made.

First we state a lemma, showing that $\preceq$ defines a total order on the set of stable vectors.

Lemma 4.3 *$\preceq$ defines a total order between all stable vectors in a given run.*

Proof: Consider two stable vectors T_1 and T_2 in run R. We show that either $T_1 \preceq T_2$ or $T_1 \succeq T_2$. By definition, a set P_1 of $n-k+1$ distinct processors broadcast T_1 and a set P_2 of $n-k+1$ distinct processors broadcast T_2. Since $n > 2(k-1)$, this implies that $P_1 \cap P_2 \neq \emptyset$. Consider some processor $p \in P_1 \cap P_2$. Since p broadcasts both T_1 and T_2, it must have broadcast one first. Suppose, without loss of generality, T_1 is broadcast before T_2. Then T_2 is computed by a series of applications of **update** and therefore $T_1 \preceq T_2$. If T_2 is broadcast first, then $T_1 \succeq T_2$. So, some processor forces the total order between any pair of stable vectors. □

Lemma 4.4 *There are at most k possible stable vectors in any run of the protocol.*

Proof: We claim two facts, which together imply the preceding statement.

1. A *stable* vector in any run is a $(k-1)$-*eligible* vector.
2. For any t, there is at most one *stable* vector in any run with *exactly* t entries with value ϕ.

To prove the first fact, note that a stable vector is broadcast by at least $n-k+1$ distinct processors. Therefore, it will contain an initial value for at least all these processors. This leaves at most $k-1$ entries with value ϕ. To prove the second fact, suppose there are two distinct stable vectors each with t entries with value ϕ. But this contradicts the fact that there is a total order between the two vectors. So, there cannot be two stable vectors with the same number of entries with value ϕ.

Now, proving the lemma is straightforward. We just note that there can be at most one stable vector with t entries with value ϕ, where t ranges from 0 to $k-1$ inclusive. That gives at most k stable vectors in any run.

□

Since there can be at most k different stable vectors, there can be at most k different decisions in any run. So, we have shown the existence of a $(k-1)$-resilient protocol for k-set agreement.

What we have presented here is really a class of algorithms which we call "stable vector" algorithms. We extend the definition of a stable vector for t-resilient protocols to be a vector, $n-t$ copies of which have been received by a processor in a run of that protocol. Each processor maintains a vector which it broadcasts to all processors and updates according to information received from other processors, in each round. Once it obtains a stable vector, it uses the COMPUTE function to decide and then terminates. By varying our COMPUTE function, we can make the algorithm satisfy different properties. In the above, we showed that we can define COMPUTE so as to satisfy agreement in the presence of $(k-1)$ faults. In the following, we prove that no k-resilient stable-vector algorithm exists for the k-set agreement problem. In other words, no COMPUTE function exists which would satisfy agreement in the presence of k faults. Note that for a k-resilient stable vector algorithm, any **k-eligible** vector may be a stable vector.

Theorem 4.5 *There exists no k-resilient "stable vector" algorithm for the k-set agreement problem.*

Proof Sketch:
Suppose a k-resilent stable-vector protocol exists for k-set agreement. Any set of k-eligible vectors $T_0, \ldots, T_k$, where for all i, $T_i \prec T_{i+1}$, could be stable vectors for a run of the protocol. Then, the decision set Y of the run is $\{\mathbf{compute}(T_i) | \, 0 \leq i \leq k\}$. Now, $|Y| \leq k$ if and only if for some i, j, $i \neq j$, $\mathbf{compute}(T_i) = \mathbf{compute}(T_j)$.

Therefore, there must exist a COMPUTE function satisfying agreement such that for any set of $k+1$ k-eligible vectors $T_0, \ldots, T_k$ where $T_i \prec T_{i+1}$, there exists $i, j, i \neq j$, such that $\mathbf{compute}(T_i) = \mathbf{compute}(T_j)$. We prove below that no such COMPUTE function exists.

This proof has the same flavor as that of the Uncertainty Theorem. To avoid repetition, we will only prove the result for the case $k = 2$ (and $V = \{a, b, c\}$) here, to illustrate the main differences.

We construct a triangle on a grid whose vertices, interior and boundary points represent 2-eligible vectors. Each vector therefore has at most 2 "blank" entries. The three vertices of the triangle represent the complete vectors (with no "blank" entries) $(a, \ldots, a)$, $(b, \ldots, b)$ and $(c, \ldots, c)$.

We now informally describe the enumeration of vectors on the grid. Figure 3 below illustrates this for $n = 4$. Every alternate row contains complete vectors (the other rows don't contain any). In these rows, complete vectors appear in every alternate point. To decide which complete vector occupies each space, notice that, in the vertical dimension, two consecutive complete vectors differ in exactly one entry; the upper vector contains an 'a' while the lower vector contains a 'b'. In the horizontal dimension, too, two consecutive complete vectors differ in exactly one entry; the left vector contains a 'b' while the right vector contains a 'c'. This places all complete vectors on the grid. The relative positioning of the complete vectors here is identical to that in the proof of the Uncertainty Theorem. The difference here is that partial vectors are placed between these complete vectors, as explained below.

Between each pair of complete vectors which differ in position i, a partial vector is inserted which is identical to both vectors except that it has a "blank" in position i. This still leaves some empty slots between partial vectors eg. $a-bc$ and $a-cc$. Here again, we insert a partial vector with a "blank" in the differing position eg. $a--c$. So, the triangle is constructed incrementally by filling in all the complete vectors in each alternate row and column. Then the partial vectors *with one "blank"* are inserted between complete vectors, and last, the partial vectors *with two "blank"s* are inserted between partial vectors with one "blank".

Given any COMPUTE function, we define the color of each point represented by vector *vec* to be the value COMPUTE(vec) $\in \{a, b, c\}$. We now show that our triangle satisfies the conditions of Sperner's Lemma. Note that the three vertices of the triangle have the "colors" a, b and c. To satisfy the validity condition, each vector appearing on the side of the triangle between *aaaa* and *bbbb* must have the color a or b. A similar statement is true for the other sides of the triangle, and so the conditions of Sperner's Lemma are satisfied.

Now, we can triangulate this triangle by bisecting each unit square on the grid *from bottom-left to top-right.* It is easy to verify that each unit triangle formed corresponds to a set of three 2-eligible vectors T_0, T_1, T_2 where $T_i \preceq T_{i+1}$. By Sperner's Lemma, there exists a unit triangle with all three colors. Therefore, the corresponding set of 2-eligible vectors is such that each vector is mapped to a different value by COMPUTE. This proves our result for the case $k = 2$. □

This impossibility result is based on a very restricted class of protocols which may not be interesting by itself. We however conjecture that the "stable-vector" model can be extended to a more general class of "full information" *message-passing* protocols for which a similar proof

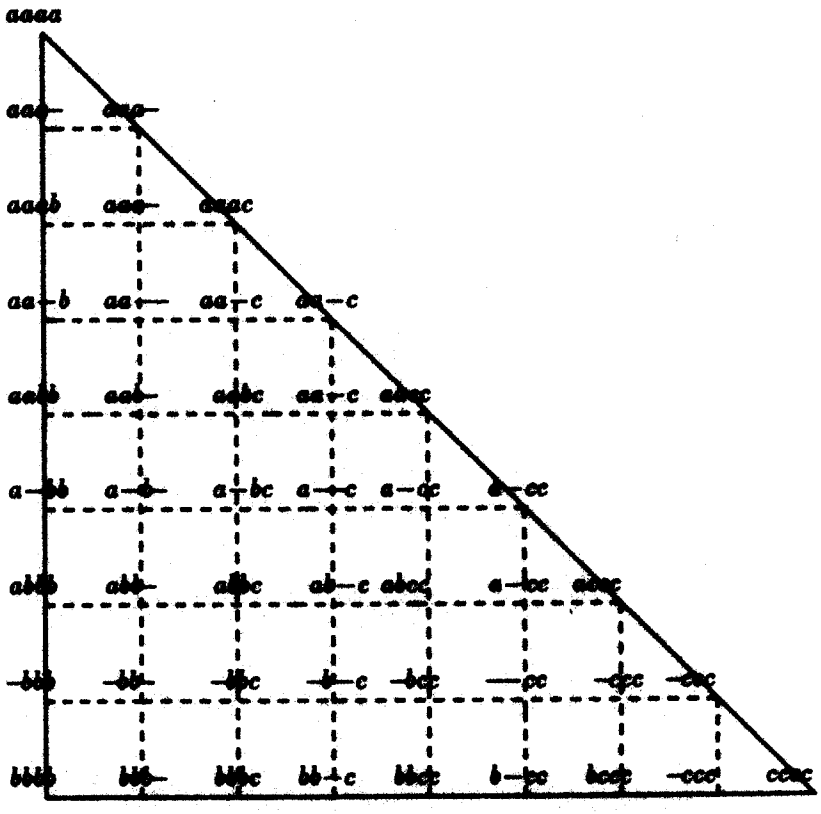

Figure 3: Enumeration of Vectors on a Grid

technique would be applicable. Attiya, Bar-noy and Dolev [ABND90] proved that any shared memory protocol could be translated into a message passing protocol. Therefore, if we can prove an impossibility result for all message passing protocols, this would extend to all shared memory protocols as well.

While the protocols we studied so far were in the *message passing* model, we can show the existence of simple $(k-1)$-resilient protocols for k-set agreement in the *shared memory (read/write)* model.

In any shared memory model, there are two parameters that can be varied. One is the number of shared memory cells we use, and the other is the number of different values that can be written into each cell.

We have the following simple $(k-1)$-resilient protocol for k-set agreement in the *shared memory (read/write)* model using 1 $(|V|+1)$-valued cell. The set of input/output values is $V = \{1, 2, \ldots, m\}$, where $m > k$.

> The memory cell is initialized to 0. Each processor p_i, for $i \in \{1, 2, \ldots, k\}$ writes its initial value x_i in the memory cell, and decides on x_i. Each processor p_j where $j > k$ reads the memory cell repeatedly until it reads a non-zero value. It then decides on that value.

Clearly, a non-zero value will be eventually written into the memory cell as long as at most $k-1$ processors die. Also, at most k values will be written. Therefore, at most k different values will be decided upon.

Now, we propose a different algorithm which uses m 2-valued memory cells, where $|V| = m$.

> The memory cells $X_1, X_2, \ldots, X_m$ are all initialized to 0. Each processor p_i, for $i \in \{1, 2, \ldots, k\}$ writes 1 into memory cell X_j, if its initial value x_i is j. It then decides on x_i. Each of the remaining processors reads all the memory cells in turn until it reads the value 1 in some memory cell m_j. It then decides the value j.

Clearly, one of the memory cells will eventually have a non-zero value, and at most k of them will. Therefore, every non-faulty processor will eventually decide and at most k values will be decided upon.

Both these protocols allow exactly k processors to make independent decisions while all other processors decide based on those decisions. This is important in satisfying k-set agreement, since it

guarantees that at most k decisions will be made. Unfortunately, the protocols are not k-resilient since if all of these k *deciding* processors die without making a decision, the other processors will not be able to decide. This seems to suggest the absence of k-resilient protocols for this problem in the shared memory model as well.

These two protocols suggest that we can get some kind of trade-off between the number of memory cells and the number of values that can be written into each memory cell. It also gives rise to a number of different models which vary in strength depending on the number of values allowed in each memory cell and the number of memory cells. We can also measure the complexity of any problem by the weakest model in which a protocol for it exists.

In this paper, we have considered only the most basic primitives for shared memory, namely *read* and *write*. The whole problem takes a new dimension if one considers strong wait-free primitives such as *read-modify-write*, *sticky bits* or *compare-and-swap*. We can compare consensus and set consensus in each of these stronger models in an attempt to get a separation between the problems. In particular, while [LAA87] showed that there is no 2-resilient consensus protocol in the *test-and-set* shared memory model, we can show the existence of a $(2k-1)$-resilient k-set consensus protocol. We conjecture that no $2k$-resilient k-set consensus protocol exists in this model.

5 Non-Triviality Conditions for Set-Consensus

In this section, we would like to discuss our motivation for choosing the *agreement* condition as our necessary non-triviality condition. Our intention was to define the k-set consensus problem so that it was resilient to $k-1$ faults *but* was not resilient to k faults. With this in mind, we study a number of different non-triviality conditions. The non-triviality condition for the canonical consensus problem was that if the initial values of all the processors are the same, that value has to be decided by all non-faulty processors. Actually, a weaker condition is sufficient for the impossibility proof of [FLP83], that some run must decide 0 and some run must decide 1. This eliminates the possibility that any default value is decided upon. Consider the following non-triviality conditions, all of which are natural extensions of the [FLP83] condition:

N1 All values $v \in V$ are decided by some processor in some run.

N2 There exist two *different* subsets of V such that each of them is the set of decided values in some run.

N3 For all processors p_i, y_i (the value decided by p_i) is not the same in every run.

N4 For all values $v \in V$, if the initial value of every processor is v in a run, v is the only possible decided value.

Unfortunately, none of these conditions are desirable non-triviality conditions for the set consensus problem. Remember that our intent was to formulate a problem which would capture some of the complexity of the renaming problem. These conditions turn out to be insufficient. We describe some trivial message-passing protocols below which solve the k-set consensus problem for $k > 1$, satisfying the above conditions.

Protocol 1:

> Given a set of processors $\{p_1, \ldots, p_n\}$, p_1 decides on its own value x_1 and broadcasts it. All other processors wait to receive from p_1 and decides on its value. If they do not hear from p_1 in one step, they decide on a default value v'.

Since p_1 can take any initial value, the different sets of decision values possible would be $\{\{v\} | v \in V\} \cup \{\{v, v'\} | v \in V\}$. This satisfies all three conditions $N1$, $N2$ and $N3$. Furthermore, the protocol is n-resilient for 2-set consensus!

The following is a t-resilient protocol for k-set consensus, where $t < n(k-1)/k$, which also satisfies condition $N4$. In particular, the protocol solves 2-set consensus as long as $t < n/2$, where t is the number of faults.

Protocol 2:

> Each processor performs a *receive* repeatedly until it receives the values of $n - t$ processors. If all the values it receives are the same, it decides on that value. Otherwise, it decides on v'.

What is wrong with these protocols? One reason they are uninteresting is that the set of possible decisions is trimmed down to size two just based on the initial values of all the processors. So, in some sense the decision is pre-determined before a single step of the protocol is taken. We can formalize this using labels by saying that each initial configuration has a label of size at most two. In *Protocol* 1, the default value, v' is in the label of *each* initial configuration. In *Protocol* 2, this is not the case for initial configurations where all processors have the same initial value. However,

for all other initial configurations, there is still a default value decided upon, and some values are eliminated independent of the actions based on the protocol.

What we need is an "uncertain" k-set consensus problem, where the set of decided values from the initial configurations does not *already* satisfy the validity condition *before* any action of the protocol. We would like the actions of the protocol to eliminate the uncertainty. So, we let the *uncertainty* condition defined in Section 2 be our new non-triviality condition.

Even the uncertainty condition fails to give any added complexity to the set consensus problem. We present a simple protocol which solves the k-set consensus problem with the *uncertainty* condition, given as many as $(n-1)$ faults.

Protocol 3:

> Each processor broadcasts its value. Processor p_1 decides on the first value it receives, and broadcasts its decision. Every other processor waits for p_1's decision. If it receives the value in a certain time, it decides on that value. Otherwise, it decides on v'.

Note that the above protocol satisfies the uncertainty condition since any initial configuration such that $|X| \geq k+1$ is $(k+1)$-valent, where X is the set of input values of the configuration. In addition, every run reaches 2-set consensus. However, the presence of a *default value* still makes the problem trivial.

We need a stronger property to make the k-set consensus problem more difficult. Our goal was to formulate a problem which is *not* resilient to k faults. The *agreement* condition is a natural property which seems to eliminate the possibility of a default value. Note that we did not need the stronger agreement condition to achieve non-triviality for the canonical consensus problem in [FLP83]. However, for set consensus, the agreement condition is imperative in achieving non-triviality.

6 Conclusion and Open Problems

The main question that we are yet to answer is if there exists a k-resilient protocol for k-set agreement, in a totally asynchronous system. We have shown that the problem does have a $(k-1)$-resilient protocol. We have also shown that any k-resilient protocol for the problem must have a $(k+1)$-valent initial configuration. [FLP83] proves impossibility of 1-resilient protocols for consensus by showing the existence of a bivalent initial configuration, and then showing that it was always

possible to create a valid run which remained in a bivalent configuration. It is unclear whether a similar inductive proof could be used to show that there exists a run in the protocol which is forever $(k+1)$-valent. So far our attempts at generalizing the inductive proof have failed. We have shown that for a restricted class of protocols (defined for the message-passing model), namely the stable-vector protocols, no k-resilient protocols exist, but have not been able to extend this result to all protocols. We still believe that there is no k-resilient protocol for the problem, but it is possible that any impossibility proof might require a totally new approach.

It would be interesting to see if the intuitive similarities between the k-set agreement problem and the renaming problem could be formalized. In other words, we would like to see if any reductions were possible from one problem to the other. The same goes for other practical problems that arise in distributed computing, like mutual exclusion. The ultimate goal would be to understand and characterize the boundary between possible and impossible in asynchronous systems with multiple faults.

7 Acknowledgments

I would like to thank Richard Ladner, whose advice and insight were critical throughout this work. Protocol 3 was suggested by Michael Fischer. Michael Saks pointed out Sperner's Lemma to me, which really simplified the proof of the impossibility result for the $k = 2$ case. I also benefitted from a number of discussions with him to come up with the general proof. I would also like to thank Hagit Attiya, Akhilesh Tyagi and Jennifer Welch for some very helpful comments and suggestions. Jennifer Welch noticed that the impossibility result was independent of the synchrony assumptions. Last, but not least, I would like to thank the referees, whose sometimes caustic criticism helped to improve the paper greatly.

I am grateful to the Department of Computer Science at the University of North Carolina at Chapel Hill for providing me with the resources and the environment to conduct this research.

References

[ABND+87] H. Attiya, A. Bar-Noy, D. Dolev, D. Koller, D. Peleg, and R. Reischuk. Achievable Cases in an Asynchronous Environment. In *IEEE Symposium on Foundations of Computer Science*. IEEE, 1987.

[ABND90] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing Memory Robustly in Message-Passing Systems. In *ACM Symposium on Principles of Distributed Computing.* ACM, 1990.

[ADG84] H. Attiya, D. Dolev, and J. Gil. Asynchronous Byzantine Consensus. In *ACM Symposium on Principles of Distributed Computing.* ACM, 1984.

[BMZ88] O. Biran, S. Moran, and S. Zaks. A Combinatorial Characterization of the Distributed Tasks Solvable in the Presence of One Faulty Processor. In *ACM Symposium on Principles of Distributed Computing.* ACM, 1988.

[BO83] M. Ben-Or. Another Advantage of Free Choice : Completely Asynchronous Agreement Protocols. In *ACM Symposium on Principles of Distributed Computing.* ACM, 1983.

[Bra85] G. Bracha. An $O(\log n)$ Expected Rounds Randomized Byzantine Generals Algorithm. In *ACM Symposium on Theory of Computing*, 1985.

[BW87] M. F. Bridgland and R. J. Watro. Fault Tolerant Decision Making in Totally Asynchronous Systems. In *ACM Symposium on Principles of Distributed Computing.* ACM, 1987.

[DDS83] D. Dolev, C. Dwork, and L. Stockmeyer. On the Minimal Synchronism Needed for Distributed Consensus. In *IEEE Symposium on Foundations of Computer Science.* IEEE, 1983. A revised version appeared in JACM 34 (January 1987).

[DLS84] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the Presence of Partial Synchrony. In *ACM Symposium on Principles of Distributed Computing.* ACM, 1984.

[FLP83] M. Fischer, N. Lynch, and M. Paterson. Impossiblity of Distributed Consensus with One Faulty Process. In *ACM Symposium on Principles of Database Systems*, 1983. A revised version appeared in JACM 32 (April 1985).

[Kuh60] H.W. Kuhn. Some Combinatorial Lemmas in Topology. *IBM Journal of Research and Development*, 4(5):518–524, 1960.

[LAA87] M. C. Loui and H. H. Abu-Amara. Memory Requirements for Agreement Among Unreliable Asynchronous Processes. *Advances in Computing Research*, 4:163–183, 1987.

[Rab83] M. Rabin. Randomized Byzantine Generals. In *IEEE Symposium on Foundations of Computer Science.* IEEE, 1983.

[Spe28] E. Sperner. Neuer Beweis für die Invarianz der Dimensionszahl und des Gebietes. *Abh. Math. Sem. Univ. Hamburg*, 6:265–272, 1928.

[TKM89] G. Taubenfeld, S. Katz, and S. Moran. Impossibility Results in the Presence of Multiple Faulty Processes. In *Proceedings of the Symposium on Foundations of Software Technology and Theoretical Computer Science*, 1989.

[Tod76] M.J. Todd. The Computation of Fixed Points and Applications. *Lecture Notes in Economics and Mathematical Systems*, 124, 1976.