Technion - Israel Institute of Technology
Computer Science Department

A Direct Proof of the Asynchronous Lower Bound for k-Set Consensus

by

Hagit Attiya

Technical Report #0930
April 1998

A Direct Proof of the Asynchronous Lower Bound for k-Set Consensus

Hagit Attiya*
Department of Computer Science
The Technion, Haifa 32000, Israel

April 23, 1998

Abstract

This paper presents a direct proof, using elementary graph theory, for the impossibility of solving k-set consensus in the presence of f crash failures using only read and write operations, when $k \leq f$.

1 Introduction

The *k-set consensus* problem [6] requires nonfaulty processors to decide on a small set of different values (of size k), so that every decision value is some processor's input. The consensus problem is a special case of this problem, with $k = 1$. The k-set consensus problem is solvable in an asynchronous system subject to crash failures using read/write operations if and only if $f < k$, where f is the number of faults to be tolerated. An algorithm was presented by Chaudhuri [6],[1] while the lower bound was proved (independently) in three papers, by Borowsky and Gafni [5], by Herlihy and Shavit [7], and by Saks and Zaharoglou [8].

All three lower bound proofs invest a great effort in translating concepts of distributed computing into some form of topology, so that some version of Sperner's Lemma could be applied; yet, there are simple proofs of Sperner's Lemma that translate the concepts of algebraic topology back into graph-theoretic notions (for example, Tompkins [9]).

This paper presents a proof of the lower bound for k-set consensus, using only elementary graph-theoretic concepts.

*Email: `hagit@cs.technion.ac.il`.
[1]The algorithm is described for message passing systems, but can be easily adapted to shared memory systems.

In order to prove the lower bound, we concentrate on a restricted set of executions. These executions, called *block executions*, were used in all other explicit proofs of the asynchronous lower bound for k-set consensus [3, 5, 8]. Section 5 discusses why the same set of executions is used in all proofs.

We represent block executions of an algorithm by an undirected graph; in the graph, each node represents a block execution, and there is an edge between two nodes if exactly one processor can distinguish between them. This graph representation allows to imitate a very simple proof of Sperner's Lemma and derive the lower bound.

Borowsky and Gafni [5] also represent block executions by graphs; however, in their representation, nodes correspond to *views* of individual processors, and executions correspond to special sub-cliques. Saks and Zaharoglou [8] create a topological space in which block executions correspond to points, and show it is compact. A more detailed comparison with other proofs of the lower bound appears in Section 6.

2 Block Executions and Their Properties

We assume a standard asynchronous shared memory model, similar to the one used, e.g., in [2]. A set of n processors, $p_1, \ldots, p_n$, communicate by reading from and writing to n shared variables, $v_1, \ldots, v_n$; only processor p_i writes to v_i, while all processors can read from v_i. There is no loss of generality in assuming this model, since other reads and write patterns can be simulated by it.

Configurations in this model are $2n$-vectors containing the states of all processors and all shared variables; in the *initial configuration*, all processors are in some initial state, and all shared variables contain $\perp$. An *event* specifies the index of a processor which gets to take a step. *Executions* are alternating sequences of configurations and events, starting with the initial configuration. In this sequence, each configuration is derived from the preceding configuration by letting the processor whose index is specified in the intermediate event take a step according to the algorithm. A processor is *nonfaulty* in an infinite execution if it takes an infinite number of steps.

The *k-set consensus* problem is formally defined as follows. Each processor p_i has special state components x_i, the *input*, and y_i, the *decision*. Initially x_i holds a value from some set of possible inputs and y_i is undefined. An assignment to y_i is irreversible. A solution to the k-consensus problem must guarantee the following, in every execution:

Termination: For every nonfaulty processor p_i, y_i is eventually assigned a value.

k-Agreement: $|\{y_i : y_i \text{ is assigned}, 1 \leq i \leq n\}| \leq k$. That is, the set of decisions made by nonfaulty processors contains at most k values.

Validity: If y_i is assigned, then $y_i \in \{x_1, \ldots, x_n\}$, for every nonfaulty processor p_i. That is, the decision of a nonfaulty processor is one of the inputs.

The lower bound proof considers only *normal* algorithms, in which each processor:

- starts with a write to its shared variable, and
- alternates between writes to its variable and collects (reads of all other processors variables), and
- includes a counter (which is incremented in every step) with each write to its shared variable.

Since we are not concerned with the complexity of algorithms, there is no loss of generality in making these assumptions; every algorithm can be modified to conform to these restrictions.

Let A be a normal algorithm solving k-set consensus in the presence of $f \leq n$ crash failures.

For the purpose of the lower bound, it also suffices to consider only a subset of all possible executions of A, called block executions, defined below. Clearly, a lower bound proved for a subset of the executions of A holds for all executions.

A *block* B is an arbitrary non-empty set of processor ids; that is, $B \subseteq \{1, \ldots, n\}$.

Consider a sequence of blocks $B_1, B_2, \ldots, B_l$, in which only the id's of processors $p_1, \ldots, p_r$ appear; it induces an *r-processor block execution* of A in which:

1. each processor p_i starts with its id i as input, for $i = 1, \ldots, r$;
2. the steps taken by processors are grouped in segments corresponding to blocks; in the ℓ-th segment corresponding to block B_ℓ, first, every processor in B_ℓ writes to its variable in an arbitrary, but fixed order, and then every processor in B_ℓ reads all variables (collects) in an arbitrary, but fixed order; and
3. each processor takes steps (as specified by the blocks, not necessarily in every one) *until it decides*, and then does not take any additional steps.

Under these restrictions, a block execution β is completely characterized by the sequence of blocks. We abuse notation and write $\beta = B_1, B_2, \ldots, B_l$.

Note that in a block execution, if a processor writes in a particular block, it also collects in the same block.

For example, Figure 1 presents a four-processor block execution β. In β, the first block is $\{p_1, p_2\}$, the second block is $\{p_2, p_3\}$, and the last block is $\{p_4\}$.

Executions in which processors take steps in blocks (Condition 2 above) were suggested in [5, 8]; they were called *immediate snapshot executions* in [5].

Let $\beta = B_1, \ldots, B_l$; the *view of processor p_i after block B_k* is p_i's state and the values of all shared variables in the configuration after the prefix of β that corresponds to $B_1, \ldots, B_k$.

$$\begin{array}{l l|l|l}
p_1: & w \quad R & & \\
p_2: & \quad\; w \quad R & w \quad R & \\
p_3: & & \quad\; w \quad R & \\
p_4: & & & w\ R
\end{array}$$

Figure 1: A block execution, $\beta = \{p_1, p_2\}, \{p_2, p_3\}, \{p_4\}$; R denotes a read of all variables.

$$\begin{array}{l l|l|l|l}
p_1: & w\ R & & & \\
p_2: & & w\ R & w \quad R & \\
p_3: & & & \quad\; w \quad R & \\
p_4: & & & & w\ R
\end{array}
\qquad\qquad
\begin{array}{l l|l|l|l}
p_1: & w \quad R & & & \\
p_2: & \quad\; w \quad R & w\ R & & \\
p_3: & & & w\ R & \\
p_4: & & & & w\ R
\end{array}$$

$\beta_1 = \{p_1\}, \{p_2\}, \{p_2, p_3\}, \{p_4\}$ $\qquad\qquad$ $\beta_2 = \{p_1, p_2\}, \{p_2\}, \{p_3\}, \{p_4\}$

Figure 2: Two block executions, $\beta_1 \overset{\neg p_1}{\sim} \beta$ and $\beta_2 \overset{\neg p_2}{\sim} \beta$.

The *view of processor p_i in β*, denoted $\beta|p_i$, is the sequence containing p_i's views after every block containing p_i; $\beta|p_i$ is empty if p_i does not appear in any block.

Two block executions, β and β', are *similar with respect to some set of processors P*, denoted $\beta \overset{P}{\sim} \beta'$, if for every processor $p_i \in P$, $\beta|p_i = \beta'|p_i$. If P includes all processors then we say that β and β' are *equivalent* and write $\beta \equiv \beta'$. In general, this does not mean that β and β' are equal; it is possible, for example, that the order of consecutive read steps is switched. However, since the order of consecutive read steps or consecutive write steps is fixed in block executions, $\beta \equiv \beta'$ implies that $\beta = \beta'$.

Of particular interest to us is the case P includes all processors but p_j, denoted $\beta \overset{\neg p_j}{\sim} \beta'$; in the non-trivial case $\beta \not\equiv \beta'$, this means that p_j is the only processor distinguishing between β and β'.

For example, consider the block executions in Figure 2. Note that p_1 distinguishes between β (from Figure 1) and β_1, but no other processor does; similarly, p_2 distinguishes between β and β_2, but no other processor does.

We say that processor p_i is *unseen* in an execution if there is no read by processor p_j, $j \neq i$, after p_i takes its first step. In a block execution β, this implies that there exists $\ell \geq 1$, such that $p_j \notin B_r$ for every $r < \ell$ and $B_r = \{p_j\}$ for every $r \geq \ell$. Intuitively, this means that p_j takes steps alone after all other processors decide; none of the other processors ever sees a step by p_j. Clearly, at most one processor is unseen in a block execution. For example, p_4 is unseen in β (Figure 1). In Appendix A, we prove a general version of the following lemma:

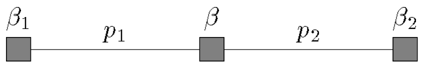


Figure 3: Part of $\mathcal{B}_4$ for the executions of Figures 2 and 1.

Lemma 2.1 (Attiya and Rajsbaum [3]) *If a processor p_j is unseen in a block execution β then there is no block execution $\beta' \not\equiv \beta$ such that $\beta \overset{\neg p_j}{\sim} \beta'$.*

If p_j is not unseen in β, then it is *seen*. For a block execution, a processor p_j is *seen in the l-th block*, if $p_j \in B_l$ and some processor $p_i \neq p_j$ is in B_r, for some $r \geq l$.

Block executions are useful since for every execution and every processor seen in it, there is a unique execution which can be distinguished only by this processor. This is formalized by the following lemma, whose proof appears in Appendix A:

Lemma 2.2 (Attiya and Rajsbaum [3]) *If p_j is seen in a block execution β, then there is a unique block execution $\beta' \not\equiv \beta$ such that $\beta' \overset{\neg p_j}{\sim} \beta$.*

3 A Graph Representation of Block Executions

We now construct a graph $\mathcal{B}_r$ whose nodes are the r-processor block executions of a fixed k-set consensus algorithm; recall that only processors $p_1, \ldots, p_r$ take steps in these executions. There is an edge between two nodes corresponding to block executions β and β' if and only if $\beta \overset{\neg p_j}{\sim} \beta'$, for some processor p_j; the edge is labeled with p_j. For example, the part of $\mathcal{B}_4$ for β, β_1 and β_2 (from Figures 1 and 2) is shown in Figure 3.

Lemma 2.2 and Lemma 2.1 imply the following simple facts about the degree of nodes:

- The edges incident to a node are labeled with distinct processors; thus, the degree of a node is at most r.
- The degree of a node corresponding to an execution without an unseen processor is exactly r.
- A node with degree $\leq r-1$ corresponds to a block execution in which some processor is unseen. Since at most one processor can be unseen in an execution, the degree of the node is exactly $r-1$.

Since the algorithm satisfies the Termination condition and since we restrict our attention to a single input, there is a finite number of r-processor block executions and therefore, the graph has a finite number of nodes.

We add an imaginary node, ϵ, to $\mathcal{B}_r$; ϵ has edges to the nodes representing executions in which some processor is unseen, labeled with the unseen processor. The extended graph is denoted $\widehat{\mathcal{B}}_r$.

In $\widehat{\mathcal{B}}_r$, the degree of all non-imaginary nodes is exactly r, and the edges incident to a non-imaginary node are labeled with r distinct processors, $p_1, \ldots, p_r$.

We now color the edges of $\widehat{\mathcal{B}}_r$. Consider an edge labeled with p_j; the edge must be adjacent to at least one non-imaginary node, corresponding to a block execution β. The edge is colored with the set of decisions of $p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_r$, i.e., all processors but p_j, in β. If this is an edge between two non-imaginary nodes corresponding to block executions β and β' then $\beta \overset{\neg p_j}{\sim} \beta'$, and therefore, all processors other than p_j decide on the same values in β and in β'. Thus, the coloring does not depend on the adjacent non-imaginary node chosen and is well-defined.

4 The Lower Bound

Assume, by way of contradiction, that A is a normal wait-free algorithm for k-set consensus among $k+1$ processors. To prove the lower bound, we show that A has a block execution in which $\{1, \ldots, k+1\}$ are decided.

For some r, $1 \leq r \leq k$, consider the graph $\widehat{\mathcal{B}}_{r+1}$ on the $(r+1)$-block executions of A. In $\widehat{\mathcal{B}}_{r+1}$, the degree of each non-imaginary node is exactly $r+1$, and each of its adjacent edges is labeled with a different processor. The *restricted degree* of a node v in $\widehat{\mathcal{B}}_{r+1}$ is the number of edges incident to v and colored with $\{1, \ldots, r\}$.

If a non-imaginary node in $\widehat{\mathcal{B}}_2$ has restricted degree 1, then one edge incident to v is colored with 1 and the other edge incident to v is colored with 2. Therefore, node v clearly corresponds to an execution in which one processor decides 1 and the other processor decides 2, i.e., an execution in which $\{1, 2\}$ are decided. The next lemma extends this observation, explaining why the restricted degree is an important notion in the lower bound proof:

Lemma 4.1 *For every r, $1 \leq r \leq k$, if the restricted degree of a non-imaginary node in $\widehat{\mathcal{B}}_{r+1}$ is exactly 1, then the node corresponds to an execution in which $\{1, \ldots, r+1\}$ are decided.*

Proof: Let β be the block execution corresponding to v, and assume the single incident edge colored with $\{1, \ldots, r\}$ is labeled by processor p_i. That is, the decisions of $\{p_1, \ldots, p_{r+1}\} \setminus \{p_i\}$ are $\{1, \ldots, r\}$.

We argue that p_i decides $r+1$ in β. Otherwise, assume p_i decides $r' \in \{1, \ldots, r\}$ in β. For every $\ell \in \{1, \ldots, r\}$ there must be a processor other than p_i that decides ℓ in β; so let p_j, $j \neq i$, be the processor that decides r' in β. Consider the edge incident to v labeled with p_j (see Figure 4); it is colored with the decisions of $\{p_1, \ldots, p_{r+1}\} \cup \{p_i\} \setminus \{p_j\}$. By the above arguments, this edge is colored with $\{1, \ldots, r\}$, implying that the restricted degree of v is 2 or more, which is a contradiction. ■

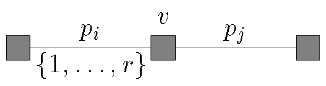


Figure 4: Illustration for Lemma 4.1.

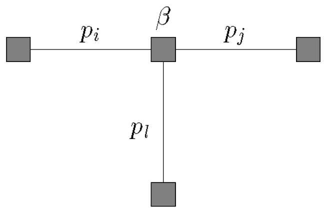


Figure 5: Illustration for Lemma 4.2.

Thus, it suffices to prove that there is a non-imaginary node in $\widehat{\mathcal{B}}_{k+1}$ whose restricted degree is 1. The next lemma shows that the restricted degree of a non-imaginary node is at most 2.

Lemma 4.2 *For every r, $1 \leq r \leq k$, the restricted degree of a non-imaginary node in $\widehat{\mathcal{B}}_{r+1}$ is at most 2.*

Proof: Assume, by way of contradiction, that there is a node v with restricted degree 3 or more, and let β be the block execution corresponding to v. This implies that v has three incident edges, say, e_i, e_j and e_l, all colored with $\{1,\ldots,r\}$; the edges are labeled by three different processors, say, p_i, p_j and p_l, respectively (see Figure 5).

What are the decisions of p_i, p_j and p_l in β?

Since e_j is colored with $\{1,\ldots,r\}$, it follows that the decisions of $\{p_1,\ldots,p_{r+1}\} \setminus \{p_j\}$ are $\{1,\ldots,r\}$. Therefore, p_l decides on some value $r' \in \{1,\ldots,r\}$. Together with the fact that e_l is colored with $\{1,\ldots,r\}$, this implies that p_j decides on the same value, r'. But then, r' appears twice in the set coloring the edge e_i, implying that e_i cannot be colored with $\{1,\ldots,r\}$. This is a contradiction. ■

Thus, it suffices to prove the following lemma:

Lemma 4.3 *For every $k \geq 1$, there is an odd number of non-imaginary nodes in $\widehat{\mathcal{B}}_{k+1}$ with odd restricted degree.*

Proof: We prove the lemma by showing it holds in $\widehat{\mathcal{B}}_{r+1}$, by induction on r, $1 \leq r \leq k$.

Base case, $r = 1$. In $\widehat{\mathcal{B}}_2$, the color of an edge is a single value, and the restricted degree of a node v is the number of edges incident to v and colored with $\{1\}$.

In $\widehat{\mathcal{B}}_2$, there are two edges incident to the imaginary node, ϵ: One edge is labeled with p_2 and is incident to the non-imaginary node corresponding to the unique execution in which p_1 takes steps on its own until deciding and then p_2 takes steps on its own. This edge is colored with the decision of p_1 which, by the Validity condition, must be p_1's input, 1. Similarly, the other edge is labeled with p_1 and is colored with 2. Thus, the restricted degree of ϵ is 1.

The sum of the restricted degrees of all nodes must be even, since each edge is counted twice in the summation. (Here we rely on $\widehat{\mathcal{B}}_2$ being finite.) Thus, there is an odd number of non-imaginary nodes in $\widehat{\mathcal{B}}_2$ with odd restricted degree.

Inductive step. Assume we have proved that there is an odd number of non-imaginary nodes in $\widehat{\mathcal{B}}_r$ with odd restricted degree, $2 \le r \le k$. We now prove the claim for $\widehat{\mathcal{B}}_{r+1}$.

Although we care about the restricted degree of non-imaginary nodes, we first consider the imaginary node, ϵ. The restricted degree of ϵ is the number of block executions with an unseen processor in which $\{1, \dots, r\}$ are decided by the seen processors. We use the induction hypothesis to prove:

Claim 4.4 *There is an odd number of* $(r+1)$*-processor block executions with an unseen processor in which* $\{1, \dots, r\}$ *are decided by the seen processors.*

Proof: Recall that only $p_1, \dots, p_{r+1}$ take steps in $(r+1)$-processor block executions. If the unseen processor is p_i, $i \neq r+1$, then the other processors cannot be sure that p_i has input i. Therefore, the other processors cannot decide on i if p_i is unseen, since A satisfies the Validity condition. (Here we are using the fact that A should work correctly for all possible assignments of input values.) That is, $\{1, \dots, r\}$ can be decided by the seen processors in an execution with an unseen processor, only if the unseen processor is p_{r+1}.

Therefore, in the $(r+1)$-processor block executions with an unseen processor in which $\{1, \dots, r\}$ are decided by the seen processors, $p_1, \dots, p_r$ run on their own, and p_{r+1} takes steps only after they decide. By removing the tail of p_{r+1}-only blocks, we get a bijection of these executions onto the r-processor block executions of A, in which only $p_1, \dots, p_r$ take steps and decide on $\{1, \dots, r\}$.

By the induction hypothesis, there is an odd number of non-imaginary nodes with odd restricted degree in $\widehat{\mathcal{B}}_r$. By Lemma 4.1 and Lemma 4.2, there is an odd number of r-processor block executions of A in which $\{1, \dots, r\}$ are decided. The claim follows from the above bijection between these executions and the $(r+1)$-processor executions in which p_{r+1} is unseen and $p_1, \dots, p_r$ decide on $\{1, \dots, r\}$. ■

Therefore, the restricted degree of ϵ is odd. The sum of the restricted degrees of nodes in $\widehat{\mathcal{B}}_{r+1}$ (including ϵ) is even, since each edge is counted twice. (Again, we rely on $\widehat{\mathcal{B}}_{r+1}$ being finite.) Therefore, there must be an odd number of non-imaginary nodes with odd restricted degree in $\widehat{\mathcal{B}}_{r+1}$, completing the proof of the inductive step. ■

By Lemma 4.2, the restricted degree of a node is at most 2, and therefore, an odd number of non-imaginary nodes in $\widehat{\mathcal{B}}_{k+1}$ have restricted degree 1. By Lemma 4.1, these nodes correspond to executions in which $\{1, \ldots, k+1\}$ are decided. Therefore, there is at least one execution of A in which $\{1, \ldots, k+1\}$ are decided, contradicting the assumption that A satisfies the k-Agreement condition, and proving our main result:

Theorem 4.5 *There is no wait-free algorithm for solving the k-set consensus problem in an asynchronous shared memory system with $k+1$ processors.*

The simulation of Borowsky and Gafni [5] implies that the lower bound can be extended to any number of failures $f \geq k$, and not just the wait-free case, $f = n-1$.

5 Why Block Executions?

All explicit proofs of the lower bound for k-set consensus [3, 5, 7] use essentially the same set of executions as we do, called here block executions. A careful examination of the proof reveals that no specific properties of block executions, except Lemma 2.1 and Lemma 2.2, were used. Any set of executions for which these lemmas hold could have been used instead of block executions.

We now show that Lemma 2.2 holds *only* for block executions, if we restrict our attention to normal algorithms (satisfying the restrictions at the beginning of Section 2). On the other hand, Lemma 2.1 holds for every execution if we assume processors halt only after read steps (see Appendix A).

An execution can be partitioned into *stretches* of read or write operations; a *read stretch* is a maximal consecutive sequence of read events, while a *write stretch* is a maximal consecutive sequence of write events.

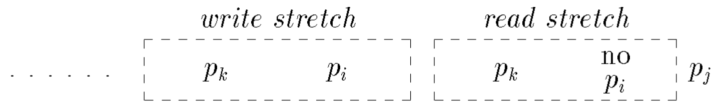
In a block execution, the processors appearing in some read stretch are exactly the processors appearing in the write stretch immediately preceding it.

In our definition of block executions, we have required the events to appear in an arbitrary, but fixed order inside the stretches. However, if α is an execution, and α' is an execution in which events within stretches are permuted, then clearly, $\alpha \equiv \alpha'$.[2] A block execution, as defined earlier, is a representative of an equivalence class of executions.

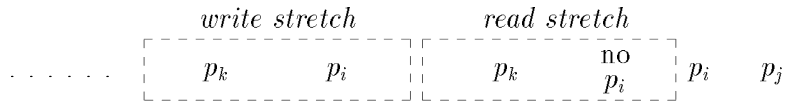
We prove the following theorem:

Theorem 5.1 *If α is not a block execution, then for some nonempty prefix α' of α, there are two executions α_1 and α_2 and a processor p_i, such that $\alpha_1 \overset{\neg p_i}{\sim} \alpha'$ and $\alpha_2 \overset{\neg p_i}{\sim} \alpha'$.*
Moreover, $\alpha_1 \not\equiv \alpha'$, $\alpha_2 \not\equiv \alpha'$, and $\alpha_1 \not\equiv \alpha_2$,

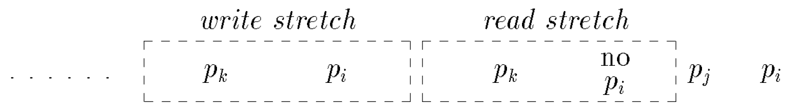
[2] To make the notions of similarity and equivalence precise, the definition of a processor's *view* should be extended to hold for an arbitrary execution; this is straightforward.



(a) α' is an extension of α'' with a step of p_j.



(b) α_1.



(c) α_2.

Figure 6: Case 3 in the proof of Theorem 5.1.

Proof: Let α' be the shortest prefix of α which cannot be extended to a block execution; clearly, α' is not empty. Assume α' is an extension of some prefix α'' with a step of some processor, p_j. By the minimality of α', α'' can be extended to a block execution; moreover, α'' is not empty.

If p_j reads in the step after α'', then there is at least one write step of p_j in α'', since the algorithm is normal; assume the last write step of p_j appears in the ℓ-th write stretch of α''. If this is the last write stretch of α'' then α' can be extended to a block execution. Otherwise, since p_j does not take any further steps in α'', p_j does not appear in the ℓ-th read stretch in α''. Therefore, α'' cannot be extended to a block execution, which contradicts the minimality of α'.

If p_j writes in the step after α'', and α'' ends with a write stretch, then α' can be extended to a block execution, since α'' can be extended to a block execution,

The remaining case is when p_j writes in the step after α'', and α'' ends with a read stretch; α' cannot be extended to a block execution only if there is a processor p_i, appearing in the last write stretch of α'', but not appearing in the last read stretch of α''. (Consider Figure 6(a).)

Let α_1 be the execution obtained from α' by inserting a read step of p_i before the write step of p_j (Figure 6(b)); let α_2 be the execution obtained by extending α' with a read step of p_i, after the write step of p_j (Figure 6(c)). Clearly, p_i has different views in α', α_1 and α_2; this implies $\alpha_1 \not\equiv \alpha_2$, $\alpha_1 \not\equiv \alpha'$, $\alpha_1 \not\equiv \alpha'$. It is also obvious that only p_i has different views in α', α_1 and α_2; that is, $\alpha_1 \overset{\neg p_i}{\sim} \alpha'$, and $\alpha_2 \overset{\neg p_i}{\sim} \alpha'$. ■

This theorem implies that if α is not a block execution then it has a prefix for which Lemma 2.2 does not hold.

6 Discussion

As mentioned before, restricting attention to executions in which processors take steps in blocks was done by both Borowsky and Gafni [5] and by Saks and Zaharoglou [8]. Borowsky and Gafni [5] give a graph representation in which each node corresponds to the view of a single processor in a specific block execution, and there are edges between views that appear in the same execution; then, they show that this graph satisfies the assumptions required for applying Sperner's Lemma, and derive the lower bound. Saks and Zaharoglou [8] represent each block execution by a point, and show that an appropriate topology of them induces a compact regular space; then, they imitate the proof of Brouwer's fixed point theorem (an extension of Sperner's Lemma) in this space and derive the lower bound.

Lemma 2.1 and Lemma 2.2 are from Attiya and Rajsbaum [3]. They use these lemmas (and additional ones) to give a topological representation of block executions; in this representation, each node corresponds to the view of a single processor in a specific block execution, and there is a simplex with certain vertices if the corresponding views appear in the same execution.[3] They show that within this topological representation, block executions induce a structure to which Sperner's Lemma applies and derive the lower bound for k-set consensus.

Herlihy and Shavit [7] derive the lower bound from a more general result characterizing what can be solved in asynchronous systems, using only read and write operations. They consider all possible executions of a full-information, normal algorithm, and represent them as a complex. In this complex, each vertex corresponds to a view of a single processor in some execution, and there is a simplex with certain vertices if the corresponding views appear in the same execution. Herlihy and Shavit show that this complex includes a subdivided simplex to which Sperner's Lemma can be applied. Herlihy and Shavit only prove that the desired subdivided simplex exists. This may seem more general than showing that specific executions induce certain structures, but Theorem 5.1 hints that this subdivided simplex must be induced by block executions, up to equivalence.

The proof presented here avoids the effort of translating concepts of distributed computing into some form of topology by a careful combination of ideas and techniques.

We restrict our attention to a block executions (like Borowsky and Gafni, Saks and Zaharoglou, and Attiya and Rajsbaum), and represent block executions by ordinary graphs (like Borowsky and Gafni), in which each node represents a block execution (like Saks and Zaharoglou). Then we imitate a very simple proof of Sperner's Lemma, which was given by Tompkins [9] (see also Bondy and Murty [4, Pages 21–23]). This shows that the *conclusion* of Sperner's Lemma holds in the domain of distributed computing, while cutting away the more complicated parts of previous lower bound proofs.

The lower bound proof presented here raises the hope that wait-free algorithms can be understood using only elementary mathematical techniques. It seems plausible that graph-

[3]The graph $\mathcal{B}_n$ is, in a sense, the "dual" of this complex.

theoretic methods similar to those used in this paper can be used to prove the lower bound for *renaming* [1]; existing proofs [3, 7] rely on the (essentially) algebraic concept of *orientation*.

Acknowledgments: This proof was developed while writing a book with Jennifer Welch; her feedback and encouragement are gratefully acknowledged. Sergio Rajsbaum taught me a lot about algebraic topology, and in particular introduced me to Tompkins' proof. Comments from Jennifer Welch, Jenny Walter and Oded Goldreich improved the paper.

References

[1] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. *J. ACM*, 37(3):524–548, July 1990.

[2] H. Attiya, N. Lynch, and N. Shavit. Are wait-free algorithms fast? *J. ACM*, 41(4):725–763, July 1994.

[3] H. Attiya and S. Rajsbaum. The combinatorial structure of wait-free solvable tasks. In *Proceedings of the 10th International Workshop on Distributed Algorithms*, number 1151 in Lecture Notes in Computer Science, pages 321–343. Springer-Verlag, 1996. Also Technical Report #CS0924, Department of Computer Science, Technion, December 1997.

[4] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* MacMillan, London and Basingstoke, 1976.

[5] E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 91–100, 1993.

[6] S. Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 103(1):132–158, July 1993.

[7] M. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 111–120, 1993.

[8] M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 101–110, 1993. Accepted to SIAM Journal on Computing.

[9] C. B. Tompkins. Sperner's lemma and some extensions. In E. F. Beckenbach, editor, *Applied Combinatorial Mathematics*, chapter 15, pages 416–455. John Wiley and Sons, Inc., New York, 1964.

A Additional Proofs

A direct proof of Lemma 2.1 appears in [3]; we prove an slight extension of the lemma, which applies to every execution of a normal algorithm (see Section 2), in which processors halt (decide or fail) only after read steps.

Lemma A.1 *If a processor p_j is unseen in an execution α then there is no execution $\alpha' \not\equiv \alpha$ such that $\alpha \overset{\neg p_j}{\sim} \alpha'$.*

Sketch of proof: If p_j is unseen in α, there is no step by another processor after p_j takes its first step (which must be a write). Assume, by way of contradiction, that there is an execution $\alpha' \not\equiv \alpha$ such that $\alpha \overset{\neg p_j}{\sim} \alpha'$. Partition α and α' into read and write stretches. Since $\alpha \not\equiv \alpha'$ there must differ in the set of processors appearing in some stretch; let ℓ be the first such stretch.

If the ℓ-th stretch in α includes the first step of p_j, then it must be a write stretch; since p_j is unseen in α, the ℓ-th stretch and all following stretches include only steps of p_j. Thus, the ℓ-th stretch must include a write step by some other processor p_i. Since p_i does not halt after a write step, there is a later read of p_i in α'. Therefore, p_i distinguishes between α and α'.

Otherwise, the ℓ-th stretch in α does not include a step by p_j. Thus there must be another processor p_i, $i \neq j$, which appears in the ℓ-th stretch of α but not in the ℓ-th stretch of α', or vice versa. It can be shown that in this case, p_i distinguishes between α and α'. ■

For completeness, we include the proof of Lemma 2.2 from [3]

Lemma 2.2 *If p_j is seen in a block execution β, then there is a unique block execution $\beta' \neq \beta$ such that $\beta' \overset{\neg p_j}{\sim} \beta$.*

Proof: Assume that $\beta = B_1, B_2, \ldots$. If p_j is seen in β, then p_j is *last seen in the k-th block*, if k is the block with the largest index in which p_j is seen. It is possible that p_j appears in blocks later than B_k, but in this case β can be written as $\beta = B_1, \ldots, B_t, \{p_j\}, \ldots, \{p_j\}$, where p_j does not appear in B_l, for every l, $k < l \leq t$.

Define a block execution β' as follows:

1. If $B_k \neq \{p_j\}$, then take p_j into an earlier block by itself; that is,

$$\beta' = B_1, \ldots, B_{k-1}, \{p_j\}, B_k - \{p_j\}, B_{k+1}, \ldots, B_t, \{p_j\}, \ldots, \{p_j\} \ ,$$

   where the number of final blocks consisting only of p_j is sufficient for p_j to decide.

2. If $B_k = \{p_j\}$, then merge p_j with the next block; that is,

$$\beta' = B_1, \ldots, B_{k-1}, \{p_j\} \cup B_{k+1}, B_{k+2}, \ldots, B_t, \{p_j\}, \ldots, \{p_j\} \ ,$$

   where the number of final blocks consisting only of p_j is sufficient for p_j to decide.

Clearly, $\beta' \overset{\neg p_j}{\sim} \beta$. It can easily be shown that p_j distinguishes between β and β', and therefore, $\beta \neq \beta'$. We now show that β' is unique.

If $B_k \neq \{p_j\}$ (Case (1)), then clearly, there is another processor $p_i \in B_k$. If $B_k = \{p_j\}$ (Case (2)), then there is another processor $p_i \in B_{k+1}$: B_{k+1} is not empty and does not include p_j. Moreover, if $p_j \in B_r$, for some $r > k$, then $B_{r'} = \{p_j\}$ for every $r' \geq r$ (otherwise, p_j would be last seen in the r-th block of β).

In both cases, for every block execution β'', which is neither β nor β', p_i distinguishes between β and β'', which proves the uniqueness of β'. ■