Counting-Based Impossibility Proofs for Renaming and Set Agreement*

Hagit Attiya and Ami Paz

Department of Computer Science, Technion

Abstract. Renaming and set agreement are two fundamental sub-consensus tasks. In the *M-renaming* task, processes start with names from a large domain and must decide on distinct names in a range of size M; in the *k-set agreement* task, processes must decide on at most k of their input values. Renaming and set agreement are representatives of the classes of *colored* and *colorless* tasks, respectively.
This paper presents simple proofs for key impossibility results for wait-free computation using only read and write operations: n processes cannot solve $(n-1)$-set agreement, and, if n is a prime power, n processes cannot solve $(2n-2)$-renaming.
Both proofs consider a restricted set of executions, and combine simple operational properties of these executions with elementary counting arguments, to show the existence of an execution violating the task's requirements. This makes the proofs easier to understand, verify, and hopefully, extend.

1 Introduction

In a basic shared-memory system, n asynchronous processes communicate with each other by writing and reading from the memory. Solving a distributed *task* requires processes, each starting with an input, to decide on outputs that satisfy certain requirements. A *wait-free* algorithm for a task ensures that each process decides within a finite number of its own steps. In this model, we can only solve *sub-consensus* tasks, which are weaker than consensus but still provide some nontrivial coordination among processes. Two prime examples of sub-consensus tasks are renaming and set agreement.

In *M-renaming* [2], processes start with names from a large domain and must choose distinct names in a range of size $M \geq n$. The range of new names, i.e., the value of M, determines the efficiency of algorithms that rely on the new names [4], and hence, it is important to minimize M. Clearly, n-renaming is trivial if processes may use their identifiers, such that process p_i chooses name i; to rule out such solutions, the algorithm is required to be *symmetric* [10, 29]. There are wait-free algorithms for $(2n-1)$-renaming [2, 4, 9, 21], and it has been argued that there is no wait-free algorithm for $(2n-2)$-renaming [5, 26–28].

* This research is supported in part by Yad-HaNadiv fund and the *Israel Science Foundation* (grant number 1227/10).

In *k-set agreement* [16], processes must decide on at most k of their input values. Clearly, n processes can solve n-set agreement, by each deciding on its input value; it has been proved that $(n-1)$-set agreement cannot be solved by a wait-free algorithm [8, 27, 32].

Our contribution: This paper presents simple proofs of the lower bounds for renaming and set agreement; the proofs consider a restricted set of executions and use counting arguments to show the existence of an execution violating the task's requirements. While there are several simple proofs for set agreement [1, 3, 5], such proofs for renaming have eluded researchers for many years.

Our main result is a proof for the impossibility of $(2n-2)$-renaming, when n is a power of a prime number. The proof goes by considering the *weak symmetry breaking* (*WSB*) task: in WSB, each process outputs a single bit, so that when all processes participate, not all of them output the same bit. A $(2n-2)$-renaming algorithm easily implies a solution to WSB, by deciding 1 if the new name decided by the process is strictly smaller than n, and 0 otherwise; therefore, the impossibility of $(2n-2)$-renaming follows from the impossibility of WSB. (There is also a reduction in the opposite direction [22], but it is not needed for the lower bound.)

We prove that WSB is unsolvable when n, the number of processes, is a power of a prime number, by showing that every WSB algorithm for this number of processes has an execution in which all processes output the same value. Since we are proving a lower bound, it suffices to consider a restricted set of executions (corresponding to *immediate atomic snapshot executions* [8, 9] or *block executions* [5]). The existence of a "bad" execution, violating the task's requirements, is proved by assigning a *sign* to each execution, counting the number of bad executions by sign, and concluding that a "bad" execution exists.

Prior lower bound proofs for renaming [11–13] rely on concepts and results from combinatorial and algebraic topology. Although our proof draws on ideas from topology (see the discussion), it uses only elementary counting arguments and simple operational properties of block executions. This makes the proof easier to understand, verify, and hopefully, extend.

As a warm-up and to familiarize the reader with counting-based arguments, we prove the impossibility of wait-free solutions for $(n-1)$-set agreement, by counting the bad executions in two complementary ways. We believe this proof is interesting on its own due to its extreme simplicity.

Another intermediate result shows the impossibility of solving *strong symmetry breaking* (*SSB*)—an adaptive variant of WSB, in which at least one process outputs 1 in every execution. The impossibility proof for SSB is similar to the proof for WSB, although it is simpler—it holds for non-symmetric algorithms and for every value of n. SSB is closely related to an *adaptive* variant of renaming, in which M, the range of new names, depends only on the number of processes participating in the execution. Specifically, in $\left(2p - \lceil \frac{p}{n-1} \rceil\right)$-*adaptive renaming*, if $p < n$ processes participate in an execution, they choose distinct names in $1, \ldots, 2p-1$, and if all n processes participate they choose distinct

names in $1, \ldots, 2n-2$. This task solves SSB, by deciding on 1 if the output of $\left(2p - \lceil \frac{p}{n-1} \rceil\right)$-adaptive renaming is strictly smaller than n.

Previous research: The impossibility of $(2n-2)$-renaming was proved by considering WSB [5, 26–28]. All these papers claim that no algorithm solves WSB for any number of processes, using the same topological lemma. A few years ago, however, Castañeda and Rajsbaum [12, 13] proved that this lemma is incorrect, and gave a different proof for the impossibility of WSB, which holds only if the binomial coefficients $\binom{n}{1}, \ldots, \binom{n}{n-1}$ are not relatively prime (see also [11]). It can be shown that these values of n are precisely the prime powers, indicating that the lower bound holds for a small fraction of the possible values of n.[1] For the other values of n, Castañeda and Rajsbaum gave a non-constructive proof, using a subdivision algorithm, for the existence of a WSB algorithm [12,15]. The upper and lower bound proofs use non-trivial topological tools on *oriented manifolds*.

The k-set agreement task is *colorless*, which intuitively means that the output of one process can be adopted by another process. Colorless tasks have been extensively studied: there is a complete characterization of their wait-free solvability by simple topological conditions [27, 28], which implies that wait-free $(n-1)$-set agreement is impossible. Wait-free $(n-1)$-set agreement is also proved impossible by topological methods [5, 8, 26, 32], or graph theory [1, 3]. Set agreement and its variants have been studied in many other models as well, see, e.g., the survey in [31].

Clearly, renaming is not colorless, since the new name chosen by one process cannot be adopted by another one; such tasks are called *colored*. Little is known about colored tasks and only a few tasks were studied (e.g., [19, 29]). A good survey of renaming can be found in [14]. We hope that our more direct treatment will encourage the investigation of colored tasks.

2 Preliminaries

We use a standard model of an asynchronous shared-memory system [6]. There is a set of n *processes* $\mathbb{P} = \{p_0, \ldots, p_{n-1}\}$, each of which is a (possibly infinite) state machine. Processes communicate with each other by applying read and write *operations* to shared registers. Each process p_i has an unbounded *single-writer multi-reader register* R_i it can write to, which can be read by all processes.

An *execution* of an algorithm is a finite sequence of read and write operations by the processes. Each process p_i starts the execution with an *input value*, denoted In_i, performs a computation and then *terminates* with an *output value*, also called a *decided value*. Without loss of generality, assume that in the algorithm, a process alternates between writing its complete state to its register and reading all the registers (performing a *scan*).

[1] Asymptotically, there are $\Theta\left(\frac{N}{\log N}\right)$ primes, and $\Theta\left(\sqrt{N}\log N\right)$ powers of primes with exponent $e \geq 2$ [23, pp. 27-28] in the interval $[1, N]$. Hence, the portion of prime powers is $\Theta\left(\frac{1}{\log N} + \frac{\log N}{\sqrt{N}}\right)$, which tends to 0 as N goes to ∞.

For a set of processes P, we say α is an execution *of* P if all processes in P take steps in α, and only them; P is the *participating set* of α. Although any process may fail during the execution, we restrict our attention to executions where every participating process terminates (possibly by taking steps after all other processes terminated).

For a process p_i that terminates in an execution α, the output of p_i in α is denoted $dec(\alpha, p_i)$. For a set of processes P, $dec(\alpha, P)$ is the set of outputs of the processes in P that terminate in α, and $dec(\alpha)$ is the set of all outputs in α.

Two executions α, α' are *indistinguishable* to process p_i, denoted $\alpha \overset{p_i}{\sim} \alpha'$, if the state of p_i after both executions is identical. We write $\alpha \overset{P}{\sim} \alpha'$, if $\alpha \overset{p_i}{\sim} \alpha'$ for every process $p_i \in P$.

A *block execution* [5], or *immediate atomic snapshot execution* [8, 9], is induced by *blocks*, i.e., nonempty sets of processes. A block execution α is induced by a sequence of blocks $B_1 B_2 \cdots B_h$, if it begins with all processes in B_1 writing in an increasing order of identifiers and then performing a scan in the same order, followed by all processes of B_2 writing and then performing a scan, and so on. Since we prove impossibility results, we can restrict our attention to block executions. Note that given a block execution as a sequence of read and write operations, there is a unique sequence of blocks inducing the execution: each consecutive set of write operations determines a block.

We consider *wait-free* algorithms, in which each process terminates in a finite number of its own steps, regardless of the steps taken by other processes. Since only executions with a bounded set of inputs are considered, there is a common upper bound on the number of steps taken in all these executions.[2]

3 Impossibility of $(n-1)$-Set Agreement

The *k-set agreement* task is an extension of the *consensus* task, where processes have to decide on at most k values. Process p_i has an input value (not necessarily binary), and it has to produce an output value satisfying:

k-Agreement: At most k different values are decided.
Validity: Every decided value is an input value of a participating process.

A process p_i is *unseen* in an execution α if it takes steps in α only after all other processes terminate. In this case, α is induced by $B_1 \cdots B_h \{p_i\}\{p_i\}^*$, where $p_i \notin B_j, 1 \leq j \leq h$, and $\{p_i\}^*$ stands for a finite, nonnegative number of blocks of the form $\{p_i\}$.

A process p_i is *seen* in an execution α if it is not unseen in it. A process $p_i \in B_j$ is *seen* in B_j if p_i is not the only process in B_j, or if there is a later block with a process other than p_i. The key property of block executions that we use is captured by the next lemma (this is Lemma 3.4 in [5]).

[2] In general, wait-freedom does not necessarily imply that the executions are bounded [25]. However, with a bounded set of inputs, wait-freedom implies that there is a bound on the number of steps taken by a process in *any* execution.

Lemma 1. *Let P be a set of processes, and let $p_i \in P$. If p_i is seen in an execution α of P, then there is a unique execution α' of P such that $\alpha' \overset{P-p_i}{\sim} \alpha$ and $\alpha' \neq \alpha$. Moreover, p_i is seen in α'.*

Sketch of proof. Let α be induced by $B_1 \cdots B_h \{p_i\}^*$, and let B_ℓ be the last block in which p_i is seen.

If $B_\ell = \{p_i\}$, define the new execution α' by merging B_ℓ with the successive block $B_{\ell+1}$. That is, $\{p_i\}B_{\ell+1}$ is replaced with $\{p_i\} \cup B_{\ell+1}$ (note that $B_{\ell+1}$ does not include p_i), and all other blocks remain the same.

Otherwise, if $B_\ell \neq \{p_i\}$, define α' by splitting p_i before B_ℓ, with opposite manipulation. That is, B_ℓ is replaced with $\{p_i\}B_\ell \setminus \{p_i\}$, and all other blocks remain the same.

In both cases, it might be necessary to add singleton steps at the end of the execution to ensure p_i terminates. □

Assume, by way of contradiction, that there is a wait-free algorithm solving $(n-1)$-set agreement. Let C_m, $1 \leq m \leq n$, be the set of all executions in which only the first m processes, $p_0, \ldots, p_{m-1}$, take steps, each process p_i has an input value i, and all the values $0, \ldots, m-1$ are decided. We prove that $C_n \neq \emptyset$, i.e., there is an execution in which n values are decided.

Lemma 2. *For every m, $1 \leq m \leq n$, the size of C_m is odd.*

Proof. The proof is by induction on m. For the base case, $m = 1$, consider a solo execution of p_0. Since the algorithm is wait-free, p_0 decides in h steps, for some integer $h \geq 1$. By the validity property, p_0 decides on 0, so there is a unique execution in C_1, induced by $\{p_0\}^h$. Hence, $|C_1| = 1$.

Assume the lemma holds for some m, $1 \leq m < n$. Let X_{m+1} be the set of all tuples of the form (α, p_i), $0 \leq i \leq m$, such that α is an execution where only processes $p_0, \ldots, p_m$ take steps, and all the values $0, \ldots, m-1$ are decided by processes other than p_i; p_i decides on an arbitrary value. We show that the sizes of X_{m+1} and C_{m+1} have the same parity.

Let X'_{m+1} be the subset of X_{m+1} containing all tuples (α, p_i), such that all values $0, \ldots, m$ are decided in α; we show that the size of X'_{m+1} is equal to the size of C_{m+1}. Let (α, p_i) be a tuple in X'_{m+1}, so α is in C_{m+1}. Since $m+1$ values are decided by $m+1$ processes in α, p_i is the unique process that decides m in α, so there is no other tuple (α, p_j) in X'_{m+1} with the same execution α. For the other direction, if α is an execution in C_{m+1}, then in α, $m+1$ values are decided by $m+1$ processes, and there is a unique process p_i which decides m in α. Hence, α appears in X'_{m+1} exactly once, in the tuple (α, p_i).

We next argue that there is an even number of tuples in X_{m+1}, but not in X'_{m+1}. If (α, p_i) is such a tuple, then p_i decides $v \neq m$ in α. Since $(\alpha, p_i) \in X_{m+1}$, all values but m are decided in α by processes other then p_i, so there is a unique process $p_j \neq p_i$ that decides v in α. Thus, (α, p_i) and (α, p_j) are both in X_{m+1} but not in X'_{m+1}, and these are the only appearances of α in X_{m+1}. Therefore, there is an even number of tuples in X_{m+1} but not in X'_{m+1}, implying that the

sizes of X_{m+1} and X'_{m+1} have the same parity, and hence, the sizes of X_{m+1} and C_{m+1} have the same parity.

To complete the proof and show that the size of X_{m+1} is odd, partition the tuples (α, p_i) in X_{m+1} into three subsets, depending on whether p_i is seen in α or not:

1. p_i is **seen** in α: By Lemma 1, for each execution α in which only $p_0, \ldots, p_m$ take steps, and p_i is seen, there is a unique execution $\alpha' \neq \alpha$ with only $p_0, \ldots, p_m$ taking steps, p_i is seen, and all processes other than p_i decide on the same values. Hence, the tuples in X_{m+1} in which p_i is seen in the execution can be partitioned into disjoint pairs of the form $\{(\alpha, p_i), (\alpha', p_i)\}$, which implies that there is an even number of such tuples.
2. $\boldsymbol{i \neq m}$ and p_i is **unseen** in α: Since $i \in \{0 \ldots, m-1\}$ and all values $\{0, \ldots, m-1\}$ are decided in α by processes other then p_i, the value i is decided in α by some process p_j, $j \neq i$. But p_i is unseen in α, so p_j must have decided on i before p_i introduced i as an input value. Considering the same execution without the steps of p_i at the end, we conclude that the fact that p_j decides on i contradicts the validity property of the algorithm. Hence, there are no such tuples in X_{m+1}.
3. $\boldsymbol{i = m}$ and p_m is **unseen** in α: We show a bijection between this subset of X_{m+1} and C_m. Since p_m is unseen in α, in the beginning of α all processes but p_m take steps and decide on all values $0, \ldots, m-1$, and then p_m takes steps alone. Consider the execution $\hat{\alpha}$ induced by the same blocks, but excluding the steps of p_m at the end, and note that $\hat{\alpha}$ is in C_m. On the other hand, every $\hat{\alpha}$ in C_m can be extended to an execution α by adding singleton steps of p_m the its end, (α, p_m) is in X_{m+1} and p_m is unseen in α.
   By the induction hypothesis, the size of C_m is odd, so the bijection implies that X_{m+1} has an odd number of tuples (α, p_m) in which p_m is unseen in α.

Since the sizes of C_{m+1} and X_{m+1} have the same parity, and the latter size is odd, then so is the former. □

Taking $m = n$, we get that the size of C_n is odd, and hence, non-zero, implying that there is an execution in which all n values are decided, contradicting the $(n-1)$-agreement property.

Theorem 1. *There is no wait-free algorithm solving the $(n-1)$-set agreement task in an asynchronous shared memory system with n processes.*

4 Impossibility of Symmetry Breaking (SSB and WSB)

In weak symmetry breaking (WSB), n inputless processes should each output a single bit, satisfying:

Symmetry Breaking: If all processes output, then not all of them output the same value.

WSB is easily solvable when using the processes' identifiers: $p_0, \ldots, p_{n-2}$ output 1, and p_{n-1} outputs 0. To prevent such solutions, we assume processes does not use their identifiers, and they all run the same algorithm. This restriction is enforced by demanding that WSB algorithm is *symmetric* [5, 18, 29], formalized as follows.

Let α be an execution prefix induced by a sequence of blocks $B_1 \cdots B_h$, and let $\pi : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ be a permutation. For a block B_j, let $\pi(B_j)$ be the block $\{p_{\pi(i)}\}_{p_i \in B_j}$, and denote by $\pi(\alpha)$ the execution prefix induced by $\pi(B_1) \cdots \pi(B_h)$.

A permutation $\pi : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ is *order preserving* on a set of processes P, if for every $p_i, p_{i'} \in P$, if $i < i'$ then $\pi(i) < \pi(i')$.

Definition 1. *An algorithm A is* symmetric *if, for every execution prefix α of A by a set of processes P, and for every permutation $\pi : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ that is order preserving on P, if a process p_i decides in α, then $p_{\pi(i)}$ decides in $\pi(\alpha)$, and on the same value.*

A adaptive version of WSB is the strong symmetry breaking (SSB) task, where n inputless processes should each output a single bit, satisfying the symmetry breaking property, and the following property:

1-decision: In every execution, at least one participating process outputs 1.

SSB is unsolvable even if the processes are allowed to use their identifiers, so we do not assume that algorithms are symmetric.

We can now explain the arguments used to prove the impossibility of both symmetry breaking tasks. As in the set agreement proof, we analyze the set of executions using counting arguments. Assume, towards a contradiction, that there is an algorithm A solving the relevant task—SSB or WSB. We associate A with a *univalued signed count*, a quantity that counts the executions of A in which all processes output the same value; if the univalued signed count is nonzero, then there is an illegal execution of A. We prove that for SSB, the univalued signed count is always nonzero, whereas for WSB, it is nonzero if n is a prime power.

To show that the univalued signed count is nonzero, we derive a trimmed version of A, with the same univalued signed count. Counting the univalued signed count is easier in the trimmed version: for SSB, it is immediate from the 1-decision property, and for WSB, the symmetric nature of the algorithm implies that the same values are output in different partial executions. Together with the fact that n is a prime power, this is used to show that the univalued signed count of the trimmed algorithm is nonzero, which completes the proof.

Section 4.1 defines the *sign* of an execution and presents a way to measure the size of a set of executions. Section 4.2 shows how to trim an algorithm in a way the preserves the univalued signed count. These tools are used in Sections 4.3 and 4.4 to prove the impossibility results for SSB and WSB, respectively.

4.1 Counting Executions by Signs

The *sign of a sequence of blocks* $B_1 \cdots B_h$ is $\text{sign}(B_1 \cdots B_h) = \prod_{i=1}^{h} (-1)^{|B_i|+1}$; it is positive if and only if there is an even number of even-sized blocks. The definition is crafted to obtain Proposition 1 and Lemma 3.

The *sign of an execution* α induced by $B_1 \cdots B_h$ is $\text{sign}(\alpha) = \text{sign}(B_1 \cdots B_h)$. If two executions (possibly of different algorithms) differ only in singleton steps of a process at their end, then the difference is only in odd-sized blocks, which do not change the sign, implying their signs are equal:

Proposition 1. *If α is an execution induced by $B_1 \cdots B_h$ and $\hat{\alpha}$ is an execution induced by $B_1 \cdots B_h \{p_i\}^m$, then* $\text{sign}(\alpha) = \text{sign}(\hat{\alpha})$.

Since the indistinguishable execution constructed in the proof of Lemma 1 is created by either pulling out or merging a singleton set, it must have an opposite sign. This is stated in the next lemma, extending Lemma 1 to argue about signs.

Lemma 3. *Let P be a set of processes, and let $p_i \in P$. If p_i is seen in an execution α of P, then there is a unique execution α' of P such that $\alpha' \overset{P-p_i}{\sim} \alpha$ and $\alpha' \neq \alpha$. Moreover, p_i is seen in α', and* $\text{sign}(\alpha') = -\text{sign}(\alpha)$.

This lemma is used in an analogous way to the parity argument in the proof of Lemma 2, except that here, we sum signs instead of checking the parity of a size; as in Lemma 2, pairs of executions from Lemma 3 cancel each other.

From now on, we consider only executions by all processes. For an algorithm A and for $v = 0, 1$, the set of executions of A in which only v is decided is $C_v^A = \{\alpha \mid dec(\alpha) = \{v\}\}$. These sets are defined for any algorithm, but the symmetry breaking property implies that both sets are empty, since the executions in which all processes decide on the same value, either 0 or 1, are prohibited. To prove that no algorithm solves SSB, or solves WSB when n is a prime power, we measure C_0^A and C_1^A and show they cannot both be empty.

The *signed count* of a set of executions S is $\mu(S) = \sum_{\alpha \in S} \text{sign}(\alpha)$. Clearly, $\mu(\emptyset) = 0$, and for any two disjoint sets S, T, $\mu(S \dot{\cup} T) = \mu(S) + \mu(T)$.

The *univalued signed count* of an algorithm A is $\mu\left(C_0^A\right) + (-1)^{n-1} \cdot \mu\left(C_1^A\right)$. Note that if the univalued signed count is nonzero, then A has an execution with a single output value. (The converse is not necessarily true, but this does not matter for the impossibility result.) We next define a trimmed version of A, $\text{T}(A)$, and show how this way of measuring C_0^A and C_1^A allows to prove that the univalued signed counts of A and $\text{T}(A)$ are equal (Lemma 4).

4.2 A Trimmed Algorithm

Let A be a wait-free algorithm which produces binary outputs. By assumption, process p_i alternates between write and scan operations:

write(i) to R_i
while (1) **do**
  $\boldsymbol{v} \leftarrow$ Scan $(R_0, \ldots, R_{n-1})$
  Local$_A(\boldsymbol{v})$: [calculation on $\boldsymbol{v}$
    if *cond* **then return** x
    else write($\boldsymbol{v}$) to R_i]

We derive from A a *trimmed* algorithm, T(A), that does not claim to solve any symmetry breaking task. The code of T(A) for a process p_i is:

boolean *simulated* $= 0$
write(i) to R_i
while (1) **do**
  $\boldsymbol{v} \leftarrow$ Scan $(R_0, \ldots, R_{n-1})$
  if $\boldsymbol{v}$ *contains all processes* **then return** *simulated*
  compute Local$_A(\boldsymbol{v})$
  if A *returns* x **then return** *the same value* x
  simulated $\leftarrow 1$

In every execution of T(A), the last process to take a first step sees all other processes in its first scan, takes no simulation steps and outputs 0; hence, $C_1^{\mathrm{T}(A)} = \emptyset$.

Every execution of A with an unseen process p_i is also an execution of T(A), up to the number of singleton steps of p_i at the end of the execution. By Proposition 1, changing the number of singleton steps does not affect the sign, so counting executions with an unseen process by sign is the same for both algorithms. This is used in the proof of the next lemma:

Lemma 4. *A and* T(A) *have the same univalued signed count.*

Proof. For each of the algorithms, we define an intermediate set of tuples in a way similar to the one used in the proof of the $(n-1)$-set agreement impossibility result (Lemma 1). These tuples contain executions spanning from the univalued 0 executions to the univalued 1 executions. More precisely, consider tuples of the form (α, p_i) such that in α, all processes with identifier smaller than i decide on 1, and all processes with identifier greater than i decide on 0. As in the proof of Lemma 1, the output of p_i does not matter. For an algorithm A, let:

$$X^A = \{(\alpha, p_i) \mid dec(\alpha, \{p_0, \ldots, p_{i-1}\}) = \{1\};\ dec(\alpha, \{p_{i+1}, \ldots, p_{n-1}\}) = \{0\}\}.$$

We abuse notation and define the *signed count* of the set of pairs X^A to be $\lambda\left(X^A\right) = \sum_{i=0}^{n-1}(-1)^i \mu(\{\alpha \mid (\alpha, p_i) \in X^A\})$. Again, $\lambda(\emptyset) = 0$, and for any two disjoint sets S, T, $\lambda(S \dot\cup T) = \lambda(S) + \lambda(T)$. Similar notations are used for T(A).

The $(-1)^i$ element in $\lambda\left(X^A\right)$ is used to cancel out pairs of tuples in X^A with the same execution and processes with consecutive identifiers. This is used in the first case of the next claim:

Claim. The univalued signed count of an algorithm A equals $\lambda\left(X^A\right)$.

Proof of claim. For every tuple $(\alpha, p_i) \in X^A$ there are three possibilities:

1. $dec(\alpha) = \{0, 1\}$. If $dec(\alpha, p_i) = 1$ then

$$dec(\alpha, \{p_0, \ldots, p_{i-1}, p_i\}) = \{1\};\ dec(\alpha, \{p_{i+1}, \ldots, p_{n-1}\}) = \{0\},$$

   so (α, p_{i+1}) is also in X^A. In $\lambda\left(X^A\right)$, α appears exactly twice, for (α, p_i) and for (α, p_{i+1}), and the corresponding summands cancel each other, since $(-1)^i \operatorname{sign}(\alpha) = -(-1)^{i+1} \operatorname{sign}(\alpha)$.
   If $dec(\alpha, p_i) = 0$ then, by a similar argument, (α, p_{i-1}) is also in X^A and the appropriate summands cancel each other.
2. $dec(\alpha) = \{0\}$. Then $i = 0$, α appears in X^A once, as (α, p_0), and its sign in $\lambda\left(X^A\right)$ is $\operatorname{sign}(\alpha)$, as in $\mu\left(C_0^A\right)$.
3. $dec(\alpha) = \{1\}$. Then $i = n - 1$, α appears in X^A once, as (α, p_{n-1}), and its sign in $\lambda\left(X^A\right)$ is $(-1)^{n-1} \operatorname{sign}(\alpha)$, as in $(-1)^{n-1}\mu\left(C_1^A\right)$.

Therefore, every tuple in X^A implies either two summands in $\lambda\left(X^A\right)$ that cancel each other, or a summand that appears in $\lambda\left(X^A\right)$ and in $\mu\left(C_0^A\right) + (-1)^{n-1}\mu\left(C_1^A\right)$ with the same sign. On the other hand, every execution $\alpha \in C_0^A$ appearers in X^A in a pair (α, p_0), as discussed in the second case, and every $\alpha \in C_1^A$ appearers in X^A in a pair (α, p_{n-1}), as discussed in the third case. Hence, the sums are equal. □

It remains to show that $\lambda\left(X^A\right) = \lambda\left(X^{\mathrm{T}(A)}\right)$.

For a tuple $(\alpha, p_i) \in X^A$ such that p_i is unseen in α, consider the execution $\bar{\alpha}$ of $\mathrm{T}(W)$ with the same sequence of blocks, possibly omitting singleton steps at the end; by Proposition 1, both executions have the same sign. Moreover, all processes but p_i complete the simulation of A and output the same values as in α. Hence, $(\bar{\alpha}, p_i) \in X^{\mathrm{T}(A)}$ and the contribution of (α, p_i) to $\lambda\left(X^A\right)$ equals the contribution of $(\bar{\alpha}, p_i)$ to $\lambda\left(X^{\mathrm{T}(A)}\right)$.

The sum over tuples (α, p_i) in which p_i is seen in α is 0 in both cases: fix a process p_i, and consider all the tuples $(\alpha, p_i) \in X^A$ in which p_i is seen in α. By Lemma 3, for each α there is a unique execution $\alpha' \neq \alpha$ of A such that $\alpha \overset{\mathbb{P}-p_i}{\sim} \alpha'$, hence for each $p_j \neq p_i$, $dec(\alpha, p_j) = dec(\alpha', p_j)$, and $(\alpha', p_i) \in X^A$. Moreover, p_i is also seen in α', and $\operatorname{sign}(\alpha) = -\operatorname{sign}(\alpha')$. Hence, we can divide all these tuples into pairs, (α, p_i) and (α', p_i), so that $\operatorname{sign}(\alpha) = -\operatorname{sign}(\alpha')$, each of which cancels out in $\lambda\left(X^A\right)$. The same claim holds for $\mathrm{T}(A)$ as well. □

4.3 Impossibility of Strong Symmetry Breaking

Let S be an SSB algorithm and consider its trimmed version, $\mathrm{T}(S)$. By Lemma 4, the univalued signed count of S and $\mathrm{T}(S)$ is the same, so it suffices to prove:

Lemma 5. *The univalued signed count of* $\mathrm{T}(S)$ *is nonzero.*

Proof. In every execution of $\mathrm{T}(S)$, the last process to take a first step sees all other processes in its first step and decides on $simulated = 0$, hence $C_1^{\mathrm{T}(S)} = \emptyset$.

There is a unique execution α of $\mathrm{T}(S)$ in which all processes take their first step together, take no simulation steps and decide 0. So $\alpha \in C_0^{\mathrm{T}(S)}$, and we claim

there is no other execution in $C_0^{\mathrm{T}(S)}$. Consider another execution α' of $\mathrm{T}(S)$. In α', there is a set of processes which does not see all other processes in their first scan operation, and all of them set *simulated* $= 1$. If one of these processes sees all other processes in a later scan, it decides 1. Otherwise, all these processes decide in the simulation of S, and this simulation induces a legal execution of S. By the 1-decision property of S, at least one of them decides 1 in S, and hence in $\mathrm{T}(S)$.

Hence, $C_0^{\mathrm{T}(S)} = \{\alpha\}$, and since $C_1^{\mathrm{T}(S)} = \emptyset$, the univalued signed count of $\mathrm{T}(S)$ is $\mathrm{sign}(\alpha) = (-1)^{n+1}$. □

By Lemma 4, the univalued signed count of S is also nonzero. Hence, there is an execution of S where all processes decide, and on the same value, so the algorithm does not satisfy the symmetry breaking property.

Theorem 2. *There is no wait-free algorithm solving SSB in an asynchronous shared memory system with any number of processes.*

4.4 Impossibility of Weak Symmetry Breaking

Let W be a symmetric WSB algorithm, and consider its trimmed version $\mathrm{T}(W)$.

In order to compute the univalued signed count of $\mathrm{T}(W)$, we use the fact that W is symmetric to show that every execution α of $\mathrm{T}(W)$ where not all processes participate has a set of executions with the same outputs as in α. This is formalized by defining an equivalence relation on the executions of $\mathrm{T}(W)$ and considering the equivalence classes it induces.

For an execution α of $\mathrm{T}(W)$, induced by the blocks $B_1 \cdots B_h$, let ℓ be the index of the first block after all processes have taken a step, i.e., $\cup_{j \leq \ell} B_j = \mathbb{P}$ and $\cup_{j<\ell} B_j \neq \mathbb{P}$. P_α is the set of all processes that set the simulated variable to be 1, namely, took steps before the rest of the processes appear. Formally, $P_\alpha = \cup_{j<\ell} B_j$, and $P_\alpha = \emptyset$ if $B_1 = \mathbb{P}$; denote $\overline{P_\alpha} = \mathbb{P} \setminus P_\alpha$.

For an integer m, $0 \leq m < n$, let S_m be the set of executions α such that $|P_\alpha| = m$, i.e., in which exactly m processes take steps in W. Note that every execution α is in some set S_m, since P_α is defined for every α, and $|P_\alpha| < n$ since the last process to start does not simulate a step of W.

When $m = 0$, S_0 is the set of executions in which all processes take steps in the first block. In $\mathrm{T}(W)$, they all decide 0 and halt, so S_0 contains only the execution induced by the single block $\{p_0, \ldots, p_{n-1}\}$. Hence, $|S_0| = 1$.

Fix some $m \geq 0$ and consider two executions $\alpha, \alpha' \in S_m$, such that α is induced by $B_1 \cdots B_h$ and α' is induced by $B'_1 \cdots B'_{h'}$. Define a relation $\simeq$ on S_m, such that $\alpha \simeq \alpha'$ if and only if $h = h'$, and there is a permutation $\pi : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ that is order preserving on P_α and on $\overline{P_\alpha}$, such that $B'_j = \pi(B_j)$ for every $j \in \{1, \ldots, h\}$. It is easy to check that $\simeq$ is an equivalence relation. The *equivalence class* of an execution α is $[\alpha] = \{\alpha' \mid \alpha \simeq \alpha'\}$.

Since W is symmetric, it can be verified that the equivalence classes of $\simeq$ have the following useful properties:

Proposition 2. *If $\alpha \in S_m$ and $\alpha' \simeq \alpha$, then* $\text{sign}(\alpha) = \text{sign}(\alpha')$, *$\alpha' \in S_m$, and $dec(\alpha) = dec(\alpha')$.*

Therefore, we can denote the sign of all the executions in $[\alpha]$ by $\text{sign}([\alpha])$.

For two sets of equal sizes, P and P', note that there is a unique permutation $\pi : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ that maps P to P' and $\mathbb{P} \setminus P$ to $\mathbb{P} \setminus P'$ and is order preserving on P and on $\mathbb{P} \setminus P$. This is due to the fact that all identifiers are distinct and hence, there is a single way to map P to P' in an order-preserving manner, and a single way to map $\mathbb{P} \setminus P$ to $\mathbb{P} \setminus P'$ in such manner. This implies that π is unique, which is used in the proof of the next lemma:

Lemma 6. *For every m, $0 \leq m < n$, and for every execution $\alpha \in S_m$, the size of $[\alpha]$ is $\binom{n}{m}$.*

Proof. Denote by $\binom{\mathbb{P}}{m}$ the set of all subsets of $\mathbb{P}$ of size m. Then $\left|\binom{\mathbb{P}}{m}\right| = \binom{n}{m}$.

For $\alpha \in S_m$, define $f : [\alpha] \to \binom{\mathbb{P}}{m}$ by $f(\alpha') = P_{\alpha'}$, which is well defined by Proposition 2. We prove that f is a bijection.

Let $\alpha', \alpha'' \in [\alpha]$ satisfying $f(\alpha') = f(\alpha'')$, and assume $\alpha' = \pi(\alpha)$ and $\alpha'' = \varphi(\alpha)$, for two permutations π, φ that are order preserving on P_α and on $\overline{P_\alpha}$. Since $f(\alpha') = f(\alpha'')$, we have that $P_{\alpha'} = P_{\alpha''}$, and hence, $\overline{P_{\alpha'}} = \overline{P_{\alpha''}}$. Since there is a unique permutation that maps P_α to $P_{\alpha'}$ and $\overline{P_\alpha}$ to $\overline{P_{\alpha'}}$ in an order-preserving manner, it follows that $\pi = \varphi$, implying that $\alpha' = \alpha''$.

Let $P \in \binom{\mathbb{P}}{m}$, and denote by π the unique permutation that maps P_α to P and $\overline{P_\alpha}$ to $\mathbb{P} \setminus P$ and is order preserving on both sets. Let $\alpha' = \pi(\alpha)$, s o $\alpha' \in S_m$ and $f(\alpha') = P_{\alpha'} = P$. □

Lemma 7. *If $n = q^e$ for a prime number q and a positive integer e, then the univalued signed count of* $\text{T}(W)$ *is nonzero.*

Proof. In every execution of $\text{T}(W)$, at least one process sees all other processes in its first snapshot and decides 0. Therefore, $C_1^{\text{T}(W)} = \emptyset$ and $\mu\left(C_1^{\text{T}(W)}\right) = 0$, implying that the univalued signed count of $\text{T}(W)$ is equal to $\mu\left(C_0^{\text{T}(W)}\right)$. To show that $\mu\left(C_0^{\text{T}(W)}\right)$ is nonzero, note that $C_0^{\text{T}(W)}$ is the disjoint union of $C_0^{\text{T}(W)} \cap S_m$ for $0 \leq m \leq n-1$, since each execution is in some set S_m. Hence,

$$\mu\left(C_0^{\text{T}(W)}\right) = \sum_{m=0}^{n-1} \mu\left(C_0^{\text{T}(W)} \cap S_m\right).$$

Fix $m \geq 0$. By Proposition 2, if α is in $C_0^{\text{T}(W)} \cap S_m$ then every $\alpha' \simeq \alpha$ is also in $C_0^{\text{T}(W)} \cap S_m$. Therefore, $C_0^{\text{T}(W)} \cap S_m$ consists of complete equivalence classes, and can be rewritten as $\cup_{\alpha \in C_0^{\text{T}(W)} \cap S_m} [\alpha]$. Therefore,

$$\mu\left(C_0^{\text{T}(W)}\right) = \sum_{m=0}^{n-1} \sum_{\{[\alpha] | \alpha \in C_0^{\text{T}(W)} \cap S_m\}} \mu\left([\alpha]\right).$$

By Lemma 6, the size of each equivalence class is $\binom{n}{m}$. Also, recall that all executions in an equivalence class have the same sign. Note that S_0 contains only the execution α induced by the block $\{p_0, \ldots, p_{n-1}\}$; for this execution, $P_\alpha = \emptyset$, and its sign is $(-1)^{n+1}$. Therefore,

$$\mu\left(C_0^{\mathrm{T}(W)}\right) = (-1)^{n+1} + \sum_{m=1}^{n-1} \sum_{\{[\alpha] | \alpha \in C_0^{\mathrm{T}(W)} \cap S_m\}} \binom{n}{m} \cdot \mathrm{sign}([\alpha]).$$

The following basic result of number theory follows from Lucas' Theorem.

Claim. If q is a prime number and e, m are positive integers such that $0 < m < q^e$, then $\binom{q^e}{m} \equiv 0 \pmod{q}$.

Therefore, all summands, except the first one, are 0 mod q, and hence,

$$\mu\left(C_0^{\mathrm{T}(W)}\right) \equiv (-1)^{n+1} \not\equiv 0 \pmod{q},$$

hence, $\mu\left(C_0^{\mathrm{T}(W)}\right) \neq 0$ and the univalued signed count of $\mathrm{T}(W)$ is nonzero. □

Consider a WSB algorithm W for n processes, where n is a prime power. By Lemma 7, the univalued signed count of $\mathrm{T}(W)$ is nonzero, and by Lemma 4, the same holds for W. This implies that at least one of C_0^W and C_1^W is not empty. Thus, there is an execution of W in which only 0 or only 1 is decided, contradicting the symmetry breaking property of W.

Theorem 3. *There is no wait-free algorithm solving WSB in an asynchronous shared memory system if the number of processes is a prime power.*

5 Discussion

Understanding wait-free solvable tasks is at the heart of distributed computing. This paper suggests a new approach for studying wait-free solvability of sub-consensus tasks. The novel ingredient in our approach is in *counting* (sometimes by sign) the number of executions that violate the task's requirements, as a way to show that such executions exist. This yields simple impossibility proofs for wait-free computation using only read and write operations: n processes cannot solve $(n-1)$-set agreement or SSB, and, if n is a prime power, they cannot solve $(2n-2)$-renaming. The simplicity of the proofs, and in particular, the fact they use only elementary mathematics, should make them accessible to a wider audience and increase confidence in their correctness. We hope this better understanding of colored tasks will promote further investigation of them.

Prior approaches [3, 5, 12, 28] also consider a restricted subset of executions in which all processes decide. Additionally, some parts of our proofs are analogous to known topological proofs [13, 28]. In these proofs, simplexes represent executions, where each node represents the local view of a process. These papers

use variants of Lemma 1 to prove that the set of simplexes representing block executions induces a *manifold*.

An execution α with the view of a process p_i factored out is represented by the pair (α, p_i) in our proofs; this is the analogue of the *face* of the simplex of α that is opposite to the node of p_i. Thus, counting pairs is analogous to counting faces of a simplex, as used in a classical proof of Sperner's Lemma, and in the proof of the Index lemma [24]. Indeed, Lemma 2 is analogous to Sperner's Lemma, and Lemma 4 corresponds to the use of the Index lemma in [12, 13].

The sign of an execution is used here instead of the topological notion of defining an *orientation* on a manifold and comparing it with the orientation of each simplex; univalued signed count is the counterpart of the topological *content*. The trimmed algorithm T(W) is an algorithmic way to look at the *cone construction* of [13]. Proposition 2 is proved there in a relatively complicated manner, using i-corners or *flip* operations and paths in a subdivided simplex.

Simulations, like [7,17,20], can be used to translate the lower bounds to other models, e.g., message-passing systems or models with fewer failures. More interestingly, it should be possible to derive analogous lower bounds for other models, by showing properties similar to Lemma 3. For example, in the message-passing model, we can consider layered executions [30] and take the sign according to the number of messages sent in the layers.

Finally, and perhaps most importantly, our simpler proofs may lead to the discovery of matching upper bounds, for example, an explicit $(2n-2)$-renaming algorithm when n is not a prime power.

Acknowledgements: We would like to thank Maurice Herlihy for collaboration in earlier stages of this research, Armando Castañeda, Keren Censor-Hillel, Faith Ellen, Tal Horesh, Petr Kuznetsov, Sergio Rajsbaum, Mor Weiss and the referees for valuable comments, and Amir Shpilka for discussions on number theory.

References

1. Attiya, H.: A direct lower bound for k-set consensus. PODC '98, 314
2. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., Reischuk, R.: Renaming in an asynchronous environment. J. ACM **37** (July 1990) 524–548
3. Attiya, H., Castañeda, A.: A non-topological proof for the impossibility of k-set agreement. SSS '11, 108–119
4. Attiya, H., Fouren, A.: Polynomial and adaptive long-lived (2k-1)-renaming. DISC '00, 149–163
5. Attiya, H., Rajsbaum, S.: The combinatorial structure of wait-free solvable tasks. SIAM J. Comput. **31** (April 2002) 1286–1313
6. Attiya, H., Welch, J.: Distributed computing: fundamentals, simulations, and advanced topics. Wiley series on parallel and distributed computing. Wiley (2004)
7. Borowsky, E., Gafni, E., Lynch, N., Rajsbaum, S.: The BG distributed simulation algorithm. Distributed Computing **14** (2001) 127–146
8. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for t-resilient asynchronous computations. STOC '93, 91–100

9. Borowsky, E., Gafni, E.: Immediate atomic snapshots and fast renaming. PODC '93, 41–51
10. Castañeda, A.: A Study of the Wait-free Solvability of Weak Symmetry Breaking and Renaming. PhD thesis, Universidad Nacional Autonoma de Mexico (2010)
11. Castañeda, A., Herlihy, M., Rajsbaum, S.: An equivariance theorem with applications to renaming. LATIN '12, 133–144
12. Castañeda, A., Rajsbaum, S.: New combinatorial topology upper and lower bounds for renaming. PODC '08, 295–304
13. Castañeda, A., Rajsbaum, S.: New combinatorial topology bounds for renaming: the lower bound. Distributed Computing **22** (2010) 287–301
14. Castañeda, A., Rajsbaum, S., Raynal, M.: The renaming problem in shared memory systems: An introduction. Computer Science Review **5**(3) (2011) 229–251
15. Castañeda, A., Rajsbaum, S.: New combinatorial topology bounds for renaming: the upper bound. J. ACM **59**(1) (2012)
16. Chaudhuri, S.: More choices allow more faults: set consensus problems in totally asynchronous systems. Inf. Comput. **105**(1) (July 1993) 132–158
17. Gafni, E.: Round-by-round fault detectors: Unifying synchrony and asynchrony. PODC '98, 143–152
18. Gafni, E.: Read-write reductions. ICDCN '06, 349–354
19. Gafni, E.: The 0–1-exclusion families of tasks. OPODIS '08, 246–258
20. Gafni, E.: The extended BG-simulation and the characterization of t-resiliency. STOC '09, 85–92
21. Gafni, E., Rajsbaum, S.: Recursion in distributed computing. SSS '10, 362–376
22. Gafni, E., Rajsbaum, S., Herlihy, M.: Subconsensus tasks: Renaming is weaker than set agreement. DISC '06, 329–338
23. Hardy, G.: Ramanujan: Twelve Lectures on Subjects Suggested by His Life and Work. AMS Chelsea Publishing Series. AMS Chelsea Pub. (1999)
24. Henle, M.: A Combinatorial Introduction to Topology. Dover Books on Mathematics Series. Dover (1994)
25. Herlihy, M.: Impossibility results for asynchronous PRAM. SPAA '91, 327–336
26. Herlihy, M., Rajsbaum, S.: Algebraic spans. Math. Struct. in Comp. Sci. **10** (August 2000) 549–573
27. Herlihy, M., Shavit, N.: The asynchronous computability theorem for t-resilient tasks. STOC '93, 111–120
28. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM **46** (November 1999) 858–923
29. Imbs, D., Rajsbaum, S., Raynal, M.: The universe of symmetry breaking tasks. SIROCCO '11, 66–77
30. Moses, Y., Rajsbaum, S.: A layered analysis of consensus. SIAM J. Comput. **31**(4) (2002) 989–1021
31. Raynal, M., Travers, C.: Synchronous set agreement: a concise guided tour PRDC '06, 267–274
32. Saks, M., Zaharoglou, F.: Wait-free k-set agreement is impossible: The topology of public knowledge. SIAM J. Comput. **29**(5) (2000) 1449–1483