Counting-Based Impossibility Proofs
for Distributed Tasks

Ami Paz
Counting-Based Impossibility Proofs
for Distributed Tasks

Research Thesis

In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Ami Paz

Submitted to the Senate of the
Technion—Israel Institute of Technology

Sivan 5773 Haifa May 2013
To Tal
The research thesis was done under the supervision of Professor Hagit Attiya in the department of Computer Science.

Acknowledgments I would like to thank my advisor, Prof. Hagit Attiya, for her patient guidance and help. Hagit taught me how to conduct a productive research without getting lost in the way, and showed me the importance of the clear and accurate presentation of the results.

The generous financial help of the Israel Science Foundation (grant number 1227/10) is gratefully acknowledged.
Contents

1 Introduction 3
   1.1 Our Contributions .............................................. 4
   1.2 Literature Survey ............................................. 5

2 Model of Computation 8

3 Set Agreement 10

4 Symmetry Breaking and Renaming 14

5 Impossibility of Symmetry Breaking and Renaming 16
   5.1 Counting Executions by Signs ................................. 17
   5.2 A Trimmed Algorithm .......................................... 21
   5.3 Impossibility of SSB and Adaptive Renaming .................. 24
   5.4 Impossibility of r-ISB, WSB and Renaming .................... 25
   5.5 Renaming Lower Bound for Arbitrary Values of n .......... 28

6 Divisibility of Binomial Coefficients 29

7 Conclusions and Open Questions 31
   7.1 Relation to Previous Proofs and Techniques .................. 31
   7.2 Future Research .................................................. 33
# List of Figures

2.1 Indistinguishable executions ................................................................. 9

3.1 Proof of Lemma 2—estimating the size of $X_{m+1}$ in two ways .................... 12

5.1 Proof of Lemma 5—creating a pair of indistinguishable executions ................. 18

5.2 Trimming an algorithm—executions of A and T(A) .................................. 21

5.3 Part of the proof of Lemma 7—A and T(A) have the same signed count ........... 23
Abstract

A cornerstone result in distributed computing is the impossibility of solving consensus using only read and write operations in asynchronous systems where processes may fail. The impossibility of solving consensus led to the study of sub-consensus coordination tasks, namely tasks that are weaker than consensus.

Two archetypal sub-consensus tasks are $k$-set agreement and $M$-renaming. In $k$-set agreement, $n$ processes must decide on at most $k$ of their input values; while $n$-set agreement is trivially solvable by each process deciding on its input, $(n-1)$-set agreement is unsolvable. In $M$-renaming, processes decide on distinct names in a range of size $M$. When $M$ is only a function of the total number of processes in the system, we call the task nonadaptive renaming, while if $M$ is also a function of the number $p$ of participants in the execution, we refer to the task as adaptive renaming. For any value of $n$, $(2n-1)$-nonadaptive renaming is solvable, but surprisingly, $(2n-2)$-nonadaptive renaming is not solvable if $n$ is a prime power and solvable otherwise. For general values of $n$, the only previously known lower bound on the number of names necessary for nonadaptive renaming is $n + 1$. For adaptive renaming, $(2p-1)$-adaptive renaming is known to be solvable, while $(2p - \lceil p/n-1 \rceil)$-adaptive renaming is not solvable.

Most previous impossibility proofs for $(n-1)$-set agreement, and all previous impossibility proofs for $(2n-2)$-nonadaptive renaming, use nontrivial topological tools and notions in an innovative way. Nevertheless, the use of topological notions makes the interaction between the impossibility proofs and the operational arguments harder to understand, and makes the proofs less accessible to distributed computing researches.

We present simple proofs for the above mentioned impossibility results: $n$ processes cannot solve $(n-1)$-set agreement, and cannot solve $(2p - \lceil p/n-1 \rceil)$-adaptive renaming; if $n$ is a prime power, $n$ processes cannot solve $(2n-2)$-nonadaptive renaming. For general values of $n$, we give a lower bound for nonadaptive renaming, which is proved using a reduction between nonadaptive renaming for different numbers of processes, and using results about the distribution of prime numbers.

Our 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.
List of Symbols

- $n$: number of processes in the system
- $p$: number of processes in an execution
- $p_i$: a process
- $P$: the set of all processes: \{\(p_0, \ldots, p_{n-1}\)\}
- $P$: an arbitrary set of processes
- $P - p_i$: $P \setminus \{p_i\}$
- $\alpha, \beta$: an execution or a partial execution
- $\alpha \sim \alpha'$: two executions indistinguishable to process $p_i$
- $\alpha \sim \alpha'$: two executions indistinguishable to every process $p_i \in P$
- $\text{dec}(\alpha, P)$: the set of outputs of processes in $P$ that terminate in $\alpha$
- $\text{dec}(\alpha)$: the set of outputs of all processes that terminate in $\alpha$
- \(\binom{n}{m}\): \(\frac{n!}{m!(n-m)!}\); \(\binom{n}{m}\) = 0 if $m > n$
- $q^e$: a positive power of a prime number ($e > 0$)
- $q$: a prime number
- $\pi$: a permutation
- $B_j$: block—a nonempty set of processes
- $B^h_j$: a sequence of $h$ instances of the block $B_j$
- $B^*_j$: $B^h_j$ for some nonnegative $h$
- $\pi(B_j)$: \(\{p_{\pi(i)}\}_{i \in B_j}\)
- $\simeq$: an equivalence relation defined on executions
- $[\alpha]$: the equivalence class of execution $\alpha$, by the relation $\simeq$; formally $\{\alpha' \mid \alpha \simeq \alpha'\}$

For an execution $\alpha$ induced by $B_1 \cdots B_h$:
- $\pi(\alpha)$: the execution induced by $\pi(B_1) \cdots \pi(B_h)$
- $\text{sign}(\alpha)$: $\prod_{i=1}^{h} (-1)^{|B_i| + 1}$

For an algorithm $A$:
- $C^A_v$: \{\(\alpha \mid \text{dec}(\alpha) = \{v\}\}\}
- $X^A$: \{\((\alpha, p_i) \mid \text{dec}(\alpha, \{p_0, \ldots, p_{i-1}\}) = \{1\}; \text{dec}(\alpha, \{p_{i+1}, \ldots, p_{n-1}\}) = \{0\}\}$

univalued signed count of $A$: $\sum_{\alpha \in C^A_0} \text{sign}(\alpha) + (-1)^{n-1} \cdot \sum_{\alpha \in C^A_1} \text{sign}(\alpha)$

signed count of $A$: $\sum_{(\alpha, p_i) \in X^A} (-1)^i \text{sign}(\alpha)$
Chapter 1

Introduction

Distributed systems become more and more common in our world, from computer networks through mobile phones to multicore processors. Such systems consist of computational processes that communicate with each other; the means of communication, the relative notion of time, and the possible failure patterns may differ between one system to the other, while the common idea of communicating computational units remains the same.

One way to study the capabilities and limitations of a distributed system is to consider simple coordination tasks. In a task, each process starts with an input value, communicates with other processes, decides on a value and halts.

This thesis considers algorithms for a totally asynchronous system, where \( n \) processes communicate using a shared memory, and any subset of them may crash. These algorithms are called \textit{wait-free} algorithms.

In the \textit{k-set agreement} task [25], each process starts with an arbitrary input, and has to decide on an input value of some process, such that at most \( k \) different values are decided; in the special case where \( k = 1 \), 1-set agreements is the well-known \textit{consensus} task [31]. These tasks capture situations in which many processes have to converge to a small domain of values, e.g., many inaccurate sensors that measure the same quantity, or several computers preforming the same computation. As the value of \( k \) decreases, so is the amount of disagreement on the output of the measurement or computation.

In the \textit{M-renaming} tasks [6], each of \( n \) processes has to decide on a unique value in the range \( \{1, \ldots, M\} \). We consider two variants of this task: in \textit{M-nonadaptive renaming}, \( M \) is only a function of \( n \), the number of processes in the system; in \textit{M-adaptive renaming}, \( M \) is also a function of \( p \), the number of participating processes in the specific execution. These tasks capture situations in which processes have to allocate a limited amount of resources, such as memory cells. The fewer memory cells needed, the more efficient the algorithm using the allocated cells will be; thus, it is desirable to reduce \( M \).
A cornerstone result in the field of distributed computing is the impossibility of consensus [31]. Generalizing this result, \((n - 1)\)-set agreement for \(n\) processes was proved impossible using either topological tools [17,41,49], or using graph theoretic arguments [7]. \((2p - \lceil p/n - 1 \rceil)\)-adaptive renaming was previously proved unsolvable [17], by a reduction to another task, **Strong Symmetry Breaking** (SSB), a reduction of SSB to \((n - 1)\)-set agreement, which is not wait-free solvable.

Set agreement and adaptive renaming are impossible, even if we allow processes to run different algorithms; nonadaptive renaming, on the other hand, can be easily solved by letting process \(p_i\) decide on \(i\). We are not interested in such trivial solutions, and they are ruled out by considering algorithms in which processes may only compare their identifiers, i.e., use their relative ranks. These algorithms are sometimes called **anonymous** [11,20,34,39,41], or **rank symmetric** [22]; we use the term **symmetric algorithms** [21]. An alternative motivation to symmetry, given by Attiya et al. [6], is that the original names of the processes are taken from a large, totally ordered domain, and a process does not know in advance what are the identifiers of the other processes.

The impossibility of solving \((2n - 2)\)-nonadaptive renaming using symmetric algorithms was claimed in several papers, by considering the equivalent **Weak Symmetry Breaking** (WSB) task [11,39-41]. All these papers claim that no algorithm solves WSB for any number of processes, and they all use closely related topological lemmas. A few years ago, however, Castañeda and Rajsbaum [20,21] proved that these lemmas are 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. For all other values of \(n\), Castañeda and Rajsbaum give a non-constructive proof for the existence of a WSB algorithm, using a topological subdivision algorithm [20,22]. Both upper and lower bound proofs use nontrivial topological tools on **oriented manifolds**; the lower bound result was later reproved using arguments from algebraic topology [19]. For values of \(n\) where the binomial coefficients are relatively prime, the only lower bound known on the size of the new namespace is \(n + 1\) names [6].

### 1.1 Our Contributions

We give new impossibility proofs for the tasks discussed above. Unlike many of the prior impossibility proofs for these tasks, our proofs do not use any notions from topology, and they employ only elementary tools.

We prove the impossibility of \((n - 1)\)-set agreement using a simple counting argument. The proof considers a subset of the possible executions of an alleged algorithm, counts illegal executions in this set, and proves that such executions always exist.

For SSB, we prove directly it is unsolvable for any value of \(n\). This is done by considering a subset of the executions of an alleged algorithm, assigning a sign to each execution, and
counting illegal executions by their signs. We prove that this count is nonzero, and conclude that there exists an illegal execution of the algorithm.

Then, we define a spectrum of tasks spanning from WSB to SSB, called $r$-Intermediate symmetry breaking ($r$-ISB), and prove impossibility results for these tasks as well, using tools similar to those used for SSB. A special case of this result is that when $n$ is a prime power, WSB is unsolvable.

WSB was previously proved impossible for values of $n$ where $\binom{n}{1}, \ldots, \binom{n}{n-1}$ are not relatively prime; these values are precisely the prime powers (see Chapter 6), making both results equivalent. However, this characterization indicates that the lower bound holds only for a small fraction of the possible values of $n$: in the interval $[1, N]$ there are asymptotically $\Theta\left(\frac{N}{\log N}\right)$ primes, and $\Theta\left(\sqrt{N} \log N\right)$ powers of primes with exponent $e \geq 2$ [36, pp. 27-28]. Hence, the fraction of prime powers in this interval is $\Theta\left(\frac{1}{\log N} + \frac{\log N}{\sqrt{N}}\right)$, which tends to 0 as $N$ goes to infinity.

For arbitrary values of $n$, we prove unconditionally the impossibility of $(2n - 2n^{0.525} - 2)$-nonadaptive renaming, and, assuming a conjecture from number theory, we show $(2n - \omega(\log^2 n))$-nonadaptive renaming is also unsolvable. These bounds are proved using a reduction between instances of nonadaptive renaming for different numbers of processes, and using results about the distribution of prime numbers.

### 1.2 Literature Survey

In 1985, Fischer, Lynch and Paterson published their seminal result—impossibility of consensus in a totally asynchronous system where a single process may fail [31]. Since then, research has tried to circumvent this impossibility result, e.g., using randomized algorithms [1, 13, 15] or failure detectors [23]. The solvability of consensus with some synchrony was also studied [29], and related tasks like approximate consensus were introduced [30].

Other works studied the solvability of tasks using reductions between different tasks and synchronization primitives [35,42,50]. An important progress in this line of research was done when Herlihy introduced the consensus hierarchy [38]: A primitive or a task is in level $n$ of this hierarchy if it can implement consensus for $n$ processes, but not for $n + 1$ processes. Herlihy showed that $n$-processes consensus is universal for level $n$, in the sense that it can be used to implement any other $n$-processes task in this level of the hierarchy, and vice versa.

The consensus hierarchy gives a good mapping of primitives and tasks that can be used to solve the consensus task with two processes or more. In 1987, Attiya et al. [5,6] introduced the renaming task, which is a nontrivial task that cannot be used to solve consensus even among two processes. They showed that renaming can be solved in an asynchronous system,
even when any number of processes may crash. Shortly after, Chaudhuri [24, 25] introduced the $k$-set agreement task, which is also solvable in the same system. These articles opened a whole field of research, namely, the investigation of sub-consensus tasks. Sub-consensus tasks are weaker than consensus for any number of processes, i.e., they cannot be used to implement even 2-process consensus.

When introducing $k$-set agreement [25], Chaudhuri also proved that $k$-set agreement can be solved in a system where at most $f < k$ processes may crash. The corresponding lower bound, namely, $k$-set agreement is not solvable if $f \geq k$ processes may fail, was proved independently in three papers [17, 41, 49], all using Sperner’s lemma or its variants. To subvert the impossibility result, some variants of $k$-set agreement where introduced, e.g., set agreement with restricted inputs [4, 47].

The original variant of renaming, defined by Attiya et al. [6], was nonadaptive renaming, in which the size of the new namespace is a function of the number of processes in the system, and not of the number of participating processes in a specific execution. Another variant is adaptive renaming, where the size of the new namespace allowed in an execution may vary as a function of the actual number of processes participating in the execution. The algorithm given by Attiya et al. [6] solves $(2p - 1)$-adaptive renaming, though this is not stated explicitly in the article. A third variant is order preserving renaming, also defined in the same article [6], and later studied, i.e., by Biran et al. [16], Okun [48] and Denysyuk and Rodrigues [27]. This thesis deals with the first two variants, namely nonadaptive and adaptive renaming, and not with order preserving renaming.

Many wait-free algorithms solving $(2n - 1)$-nonadaptive renaming or $(2p - 1)$-adaptive renaming have been presented over the years. Attiya et al. gave an adaptive algorithm [6]; Borowsky and Gafni proposed a fast adaptive algorithm [18], which was later improved by Gafni and Rajsbaum [33], who gave a new perspective on it as a recursive algorithm. Moir and Anderson were the first to give a long-lived, fast and adaptive algorithm [46], which was later improved and simplified by Moir [45]. Later on, Afek and Merritt introduced a fast, adaptive algorithm [3], and Attiya and Furen gave a two adaptive polynomial algorithms, one which is long-lived [8], and another, fast $(6p - 1)$-adaptive renaming algorithm [9]. Algorithms for other variants of renaming where also suggested, like group renaming [2], and renaming under eventually limited contention [43].

There have been several attempts to prove a lower bound for the namespace needed for renaming. For adaptive renaming, a chain of reductions shows that if $(2p - 2)$-adaptive renaming is wait-free solvable then so is $(n - 1)$-set agreement [17, 32], which is known to be unsolvable [7, 10, 17, 41, 49]. For nonadaptive renaming, all articles we are aware of prove the impossibility of $(2n - 2)$-nonadaptive renaming using the equivalent weak symmetry breaking (WSB) task [34] (see also Chapter 4). WSB was claimed to be unsolvable in a series of works, starting with the pioneering work of Herlihy and Shavit [40, 41] through Herlihy and Rajsbaum [39] and to
Attiya and Rajbaum [11]. All these articles were based on closely related topological lemmas; Castañeda and Rajbaum [20] later showed that in some cases these lemmas are incorrect, and proved that WSB with \( n \) processes is unsolvable if \( \left( \begin{array}{c} n \\ 1 \end{array} \right), \ldots, \left( \begin{array}{c} n \\ n-1 \end{array} \right) \) are not relatively prime, while solvable in all other cases. By showing the above condition is equivalent to \( n \) being a prime power, we conclude that for most values of \( n \), \( (2n - 2) \)-nonadaptive renaming is solvable.

Most previous proofs for the impossibility of solving \( (n - 1) \)-set agreement, and all previous proofs for the impossibility of solving \( (2n - 2) \)-nonadaptive renaming, use topological notions. The only exception is the proof of Attiya and Castañeda [7], for the impossibility of solving \( (n - 1) \)-set agreement, which uses graph theoretical arguments. Following their work, we give several impossibility proofs, which do not use topological notions. Nevertheless, a reader familiar with previous, topological proofs, can find a comparison in Section 7.1.
Chapter 2

Model of Computation

We use a standard model of an asynchronous shared-memory system \[7,12\]. A system consists of a set of \(n\) processes denoted \(P = \{p_0, \ldots, p_{n-1}\}\), each of which is a (possibly infinite) deterministic state machine. Each process has a set of possible local states, with two nonempty subsets: initial states and final states. Processes communicate with each other by applying operations to shared registers. Each process \(p_i\) has an unbounded single-writer multi-reader register \(R_i\), and two operations it can apply to the registers—write to \(R_i\), or read any of the registers. A register starts with a default initial value, which is overwritten in the first time a process writes to it, and is never rewritten to the register.

Each state machine models a local algorithm for one process. A distributed algorithm is a collection of local algorithms, one for each process. We consider only wait-free algorithms, in which each process terminates in a finite number of its own operations, regardless of the operations taken by other processes.

An execution of a distributed algorithm is a finite sequence of read and write operations by the processes. Each process \(p_i\) starts the execution from an initial state, which may encode an input value, performs a sequence of read and write operations, and then terminates in a final state, which encodes an output value. If a process \(p_i\) terminates in an execution \(\alpha\) in a final state that encodes an output value \(v\), we say that \(p_i\) decides on \(v\) in \(\alpha\).

Since we prove impossibility results, we may assume that each process proceeds in steps, where in each step the process writes its complete state to its register, reads all the registers in the system, namely preforms a scan, and then preforms a local computation. We also assume that the algorithm is a full information algorithm, i.e., the state of a process encodes all the information it read so far.

A block is a nonempty sets of processes; let \(B_1B_2\cdots B_h\) be a finite sequence of blocks. A block execution \[11\], or immediate atomic snapshot execution \[17,18\], induced by \(B_1B_2\cdots B_h\) consists of all processes in \(B_1\) writing together and then reading together, then all processes
Two block sequences inducing executions indistinguishable to \( p_1 \) and \( p_2 \) in \( B_2 \) writing together and then reading together, and so on. To describe block executions uniquely in the standard read/write model, we assume that for \( j = 1 \) to \( h \), all processes in a block \( B_j \) write in an increasing order of identifiers and then each of them performs a scan in the same order. A scan operation triggered by a process \( p_i \) returns the process a view, a vector which contains the current states of all memory registers.

Two executions \( \alpha \) and \( \alpha' \) are indistinguishable to a 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 \( p_i \in P \). See Figure 2.1.

For a set of processes \( P \), we say \( \alpha \) is an execution by \( P \) if all processes in \( P \) take steps in \( \alpha \), and only them; \( P \) is the participating set of \( \alpha \), and any process in \( P \) is a participating process. Although any process may fail during the execution, we restrict our attention to executions in which every participating process terminates without failing, i.e., it reaches a final state at the end of the execution. A prefix of an execution is called a partial execution, and we define a partial execution by a set \( P \) of processes in a similar manner.

For an execution \( \alpha \) and a set of processes \( P \), \( \text{dec}(\alpha, P) \) denotes the set of all output values of processes in \( P \) in the execution \( \alpha \), and \( \text{dec}(\alpha) \) is the set of all outputs of processes participating in \( \alpha \).

Let \( \alpha \) be a partial execution 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 partial execution 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 \( p_i, p_j \in P \), if \( i < j \) then \( \pi(i) < \pi(j) \).

**Definition 1.** An algorithm \( A \) is symmetric if, for every partial execution \( \alpha \) 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 \( \alpha \), then \( p_{\pi(i)} \) decides in \( \pi(\alpha) \), and on the same value.

For example, a solo execution of a process running a symmetric algorithm will always terminate in the same number of steps, and with the same output value, regardless of the identity of the process.
Chapter 3

Set Agreement

The first task we consider is a generalization of consensus, called \( k \)-set agreement [25], where instead of deciding on a single input value, processes have to decide on up to \( k \) different input values. In more detail, process \( p_i \) has an input value (not necessarily binary), and it has to produce an output value satisfying the following conditions:

**\( k \)-agreement:** At most \( k \) different values are decided.

**Validity:** Every decided value is an input value of a participating process.

The \( n \)-set agreement task is trivially solved by letting each process decide on its own input. We prove that \( (n-1) \)-set agreement is not wait-free solvable; if \( k \)-set agreement is wait-free solvable for some \( k \leq n-1 \), then so is \( (n-1) \)-set agreement, so the impossibility of wait-free \( (n-1) \)-set agreement trivially implies the impossibility of wait-free \( k \)-set agreement for any \( k \leq n-1 \).

A process \( p_i \) is *unseen* in an execution \( \alpha \) if it takes steps in \( \alpha \) only after all other processes terminate. In this case, \( \alpha \) 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 a block* \( B_j, 1 \leq j \leq h \), in an execution induced by \( B_1 \cdots B_h \), if \( p_i \in B_j \) and either \( p_i \) is not the only process in \( B_j \), or there is a later block, \( B_{j'}, j' > j \), with a process other than \( p_i \); in this case we say \( p_i \) is *seen* in the corresponding execution. It can be easily verified that each participating process is seen or unseen in an execution, but not both.

The key property of block executions that we use is captured by the next lemma (this is Lemma 3.4 in [11]).

**Lemma 1.** Let \( P \) be a set of processes, and let \( p_i \in P \). If \( p_i \) is seen in an execution \( \alpha \) by \( P \), then there is a unique execution of the same algorithm, \( \alpha' \neq \alpha \) by \( P \), such that \( \alpha' \overset{P}{\sim} \alpha \). Moreover, \( p_i \) is seen in \( \alpha' \).
Sketch of proof. Let $\alpha$ be induced by $B_1 \cdots B_h \{p_i\}^*$, and let $B_\ell$ be the last block in which $p_i$ is seen.

If $B_\ell = \{p_i\}$, define the new execution $\alpha'$ by merging $B_\ell$ 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.

If $B_\ell \neq \{p_i\}$, define $\alpha'$ by splitting $p_i$ before $B_\ell$, with the opposite manipulation. That is, $B_\ell$ is replaced with $\{p_i\} (B_\ell \setminus \{p_i\})$, and all other blocks remain the same.

See also Figure 5.1 on page 18. □

We extend this lemma as Lemma 5 in Chapter 5, where it is also proved. The main technical difficulty in the proof is the uniqueness claim, which is proved using case analysis.

Lemma 1 is used in the proof of the following lemma, which is the main lemma of the current chapter. In this proof, we consider the set of all tuples of the form $(\alpha, p_i)$, where $p_i$ is fixed and $\alpha$ is an execution in which $p_i$ is seen, and use Lemma 1 to split it into pairs.

To prove $(n-1)$-set agreement is not solvable, assume by way of contradiction that there is a wait-free algorithm solving this task. Let $C_m, 1 \leq m \leq n$, be the set of all executions by the first $m$ processes, $p_0, \ldots, p_{m-1}$, where 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 by all processes in which $n$ different values are decided.

Since we consider only wait-free algorithms, in which each process terminates within a finite number of steps, and there is only a single input vector, we have a finite number of possible executions. Thus, for every $m, 1 \leq m \leq n$, the size of $C_m$ is finite, and we now prove this size is always odd.

**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, $m = 1$, $C_1$ consists of solo executions by $p_0$. Since the algorithm is wait-free, $p_0$ decides in $h$ steps, for some fixed integer $h$. By the validity property, $p_0$ decides on 0, so there is a unique execution in $C_1$, induced by a sequence of $h$ blocks of the form $\{p_0\}$. 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 $(\alpha, p_i)$, $0 \leq i \leq m$, such that $\alpha$ is an execution by the processes $p_0, \ldots, p_m$, and all $m$ values $0, \ldots, m-1$ are decided in $\alpha$ 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, by defining an intermediate set. See also Figure 3.1(a).

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

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

The sizes of \( C_{m+1} \) and \( X_{m+1} \) are equal, the sizes of \( X'_{m+1} \) and \( X_{m+1} \) have the same parity, hence the sizes of \( C_{m+1} \) and \( X_{m+1} \) have the same parity.

We complete the proof by showing that the size of \( X_{m+1} \) is odd. To do so, partition the tuples \((\alpha, p_i)\) in \( X_{m+1} \) into three disjoint subsets, depending on whether \( p_i \) is seen in \( \alpha \) or not (See also Figure 3.1(b)):

1. \( p_i \) is seen in \( \alpha \): \( \alpha \) is an execution by \( p_0, \ldots, p_m \) in which \( p_i \) is seen. By Lemma 1, there is a unique execution \( \alpha' \neq \alpha \) by \( p_0, \ldots, p_m \), satisfying \( \alpha' \sim \alpha \) for every \( j \neq i \), \( 0 \leq j \leq m \), and \( p_i \) is seen in \( \alpha' \). By the indistinguishability property, all processes other than \( p_i \) decide
on the same values in $\alpha$ and in $\alpha'$, and $(\alpha', p_i)$ is also in $X_{m+1}$. Hence, for any fixed $i$, 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. $i \neq m$ and $p_i$ is unseen in $\alpha$: Since $i \in \{0, \ldots, m-1\}$ and all values $\{0, \ldots, m-1\}$ are decided in $\alpha$ by processes other than $p_i$, the value $i$ is decided in $\alpha$ by some process $p_j$, $j \neq i$. $p_i$ is unseen in $\alpha$, so $\alpha$ is induced by a sequence of blocks $B_1 \cdots B_h \{p_i\}^*$, satisfying $p_i \notin B_\ell$ for $1 \leq \ell \leq h$; let $\hat{\alpha}$ be the execution induced by $B_1 \cdots B_h$. All processes but $p_i$ take the same steps in $\alpha$ and in $\hat{\alpha}$, so $\alpha$ and $\hat{\alpha}$ are indistinguishable by all processes other than $p_i$, which hence decide on the same values in both executions; specifically, $p_j$ decides on $i$ in $\hat{\alpha}$, whereas $p_i$ does not take steps in it. Hence, $p_j$ decides in $\hat{\alpha}$ on a value that was not an input value of any process in that execution, contradicting the validity property of the algorithm. We conclude that there are no such tuples in $X_{m+1}$.

3. $i = m$ and $p_m$ is unseen in $\alpha$: We show a bijection between this subset of $X_{m+1}$ and $C_m$. Since $p_m$ is unseen in $\alpha$, in the beginning of $\alpha$ all processes but $p_m$ take steps and decide on all values $0, \ldots, m-1$, and then $p_m$ takes steps alone. As before, 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 execution $\hat{\alpha}$ in $C_m$ can be uniquely extended to an execution $\alpha$ by adding singleton steps of $p_m$ at the end, so that $(\alpha, p_m)$ is in $X_{m+1}$ and $p_m$ is unseen in $\alpha$.

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 $(\alpha, p_m)$ in which $p_m$ is unseen in $\alpha$.

In summary, $X_{m+1}$ is the disjoint union of an even sized set, an empty set and an odd sized set, thus the size of $X_{m+1}$ is odd. Since the sizes of $C_{m+1}$ and $X_{m+1}$ have the same parity, we conclude that the size of $C_{m+1}$ is also odd, as claimed.

Taking $m = n$, we get that the size of $C_n$ is odd, and hence, nonzero. Therefore, there is an execution in which all $n$ values are decided, which contradicts the $(n-1)$-agreement property. This implies the main result of this chapter:

**Theorem 3.** There is no wait-free algorithm solving the $(n-1)$-set agreement task in an asynchronous shared memory system with $n$ processes.
Chapter 4

Symmetry Breaking and Renaming

Consensus and set agreement are tasks in which processes have to converge on a small set of values. We now turn to a different kind of tasks, in which the processes have to agree on a different value for each of them. We consider some variants of this problem, starting with the M-nonadaptive renaming task, in which each of the processes has to output a unique value satisfying:

**M-namespace:** The output value is in \{1, \ldots, M\}.

In nonadaptive renaming, the size of the new namespace is a function of the total number of processes in the system, namely \( M = M(n) \). A related task is the M-adaptive renaming, which is defined in a similar manner, but with the modification that the size of the namespace allowed in an execution is also a function of the number \( p \) of processes that participate in the execution, i.e. \( M = M(n, p) \).

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 bit.

We prove the impossibility of \((2n - 2)\)-nonadaptive renaming using a reduction to WSB. Assume an algorithm \( REN \) solves \((2n - 2)\)-nonadaptive renaming, and define a WSB algorithm for a process \( p_i \):

- Simulate \( REN \) to get an output \( newName \).
- If \( newName \geq n \), decide 0, otherwise, decide 1.
Since the new names are distinct integers in \( \{1, \ldots, 2n - 2\} \), if all processes decide, at least one of them decides on a new name smaller than \( n \), and at least one of them decide on a new name greater than or equal to \( n \). An alternative reduction is taking the parity of the new name, a reduction which is more common in the literature [34, 41]; here we prefer the former reduction, as it will be useful for the following reduction as well.

Another binary task is \textit{strong symmetry breaking (SSB)}, in which \( n \) inputless processes should each output a single bit, satisfying:

\textbf{Symmetry breaking:} If all processes output, then not all of them output the same value.

\textbf{Output-one:} In every execution, at least one process outputs 1.

\( (2p - \lceil p/n - 1 \rceil) \)-adaptive renaming is a weaker version of \( (2p - 2) \)-adaptive renaming, in the sense that any algorithm solving the later also solves the former, as \( (2p - 2) \) is never greater than \( (2p - \lceil p/n - 1 \rceil) \). If \( (2p - \lceil p/n - 1 \rceil) \)-adaptive renaming is solvable then so is SSB, using the same reduction as above: in an execution of the algorithm where at most \( p \leq n - 1 \) processes output, the new names are in \( \{1, \ldots, 2p - 1\} \). Since \( 2p - 1 \leq (n - 1) + p - 1 \), at most \( p - 1 \) processes decide on new names greater or equal to \( n \), so at least one process decides on 1 in the reduction. In an execution where all processes output, symmetry breaking is achieved and 1 is decided, as argued for WSB.

Hence, to prove the impossibility of \( (2p - \lceil p/n - 1 \rceil) \)-adaptive renaming, we prove the impossibility of SSB.

We also define a class of tasks in between WSB and SSB, which we call \textit{r-intermediate symmetry breaking (r-ISB)}, in which \( n \) inputless processes should each output a single bit, satisfying:

\textbf{Symmetry breaking:} If all processes output, then not all of them output the same value.

\textbf{r-output-one:} If \( p \geq r \) processes participate in an execution, at least one of them outputs 1.

In WSB, only if \( n \) processes participate one of them has to output 1, so WSB is the same as \( n \)-ISB. In SSB, on the other hand, even if only 1 process participates it has to decide on 1, hence, SSB is the same as 1-ISB.
Chapter 5

Impossibility of Symmetry Breaking and Renaming

In this chapter, we present the main results of the thesis: counting-based impossibility proofs for the symmetry breaking tasks, which imply impossibility results for renaming. As in the set agreement impossibility 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, WSB, or r-ISB. We associate A with a univalued signed count, a quantity that counts the executions of A in which all processes output the same value; clearly, 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 r-ISB, it is nonzero if there is a prime power $q^e \geq r$ that divides $n$. Since WSB and $n$-ISB are equivalent, it follows that if $n$ is a prime power then the univalued signed count of a WSB algorithm for $n$ processes is nonzero.

To show that the univalued signed count of A is nonzero, we derive a trimmed version of A, and prove that it has the same univalued signed count as A. While the univalued signed count is the same in $A$ and in its trimmed version, evaluating the univalued signed count is easier in the trimmed version as it has more structured executions. For SSB, the univalued signed count of the trimmed version is easily proved to be nonzero from the output-one property of the SSB algorithm. For r-ISB, the symmetric nature of the algorithm implies that the same values are output in different partial executions; this is used to show that the univalued signed count of the trimmed algorithm is nonzero modulo $q$ and hence nonzero, which completes the proof.

Section 5.1 defines the sign of an execution, which is then used to define the univalued signed count of an algorithm. Section 5.2 shows how to trim an algorithm, while preserving the univalued signed count. These tools are used in Sections 5.3 and 5.4 to prove impossibility results for SSB and symmetric r-ISB, respectively. At the end of Section 5.3 we deduce the impossibility of $(2p - \lceil p/n-1 \rceil)$-adaptive renaming from that of SSB; at the end of Section 5.4
we show that solving WSB is impossible whenever $n$ is a prime power, and then use this result to deduce the impossibility of $(2n-2)$-nonadaptive renaming when $n$ is a prime power. In Section 5.5, we use the impossibility of $(2n-2)$-nonadaptive renaming to prove a weaker lower bound for nonadaptive renaming, holding for any value of $n$.

5.1 Counting Executions by Signs

A main ingredient of the proofs presented in this chapter is a more involved counting of the executions under consideration. To do this, we assign each execution with a sign, $+1$ or $-1$, crafted so as to obtain Proposition 4 and Lemma 5 below.

**Definition 2.** Let $\alpha$ be an execution induced by a sequence of blocks $B_1 \cdots B_h$. The sign of $\alpha$ is defined to be $\text{sign}(\alpha) = \prod_{i=1}^{h} (-1)^{|B_i|} + 1$.

From this definition, it is easy to deduce

$$\text{sign}(\alpha) = \begin{cases} +1 & \text{if } \alpha \text{ has an even number of even-sized blocks} \\ -1 & \text{if } \alpha \text{ has an odd number of even-sized blocks.} \end{cases}$$

Hence, odd-sized blocks do not affect the sign, and in particular, if two executions (possibly of different algorithms) differ only in singleton steps of a process at their end, then their signs are equal:

**Proposition 4.** If $\alpha$ 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 $\text{sign}(\alpha) = \text{sign}(\hat{\alpha})$.

The following lemma extends Lemma 1 to argue about signs. Both Lemma 1 and the proof of this lemma follow [11, Lemma 3.4].

The lemma is used in an analogous way to the parity argument in the proof of Lemma 2, except that here, we sum the signs of executions in a set, instead of checking the parity of the size of this set; as in Lemma 2, pairs of executions constructed by Lemma 5 cancel each other.

**Lemma 5.** Let $P$ be a set of processes, and let $p_i \in P$. If $p_i$ is seen in an execution $\alpha$ by $P$, then there is a unique execution of the same algorithm, $\alpha' \neq \alpha$ by $P$, such that $\alpha' \sim^{P\setminus p_i} \alpha$. Moreover, $p_i$ is seen in $\alpha'$, and $\text{sign}(\alpha') = -\text{sign}(\alpha)$.

**Proof.** Let $\alpha$ be an execution induced by $B_1 \cdots B_h \{p_i\}^*, B_h \neq \{p_i\}$, in which $p_i$ is seen. Denote by $B_l$ be the last block in which $p_i$ is seen.
If $B_\ell = \{p_i\}$, then since $p_i$ is seen in $B_\ell$, we have that $\ell + 1 \leq h$. Define the new blocks $B'_1, \ldots, B'_h$ by merging $B_\ell$ into its successive block:

$$B'_j = \begin{cases} 
B_j & \text{if } j < \ell \\
\{p_i\} \cup B_{\ell+1} & \text{if } j = \ell \\
B_{j+1} & \text{if } j > \ell,
\end{cases}$$

and letting $h' = h - 1$.

If $B_\ell \neq \{p_i\}$, define the new blocks $B'_1, \ldots, B'_h'$ by splitting $\{p_i\}$ before $B_\ell$:

$$B'_j = \begin{cases} 
B_j & \text{if } j < \ell \\
\{p_i\} & \text{if } j = \ell \\
B_\ell \setminus \{p_i\} & \text{if } j = \ell + 1 \\
B_{j-1} & \text{if } j > \ell + 1,
\end{cases}$$

and letting $h' = h + 1$. This construction is depicted in Figure 5.1.

In both cases, we define a new execution $\alpha'$ as the execution induced by $B'_1 \cdots B'_h' \{p_i\}^*$, where at the end of $\alpha'$, the process $p_i$ takes steps by itself until it terminates.

The blocks of $\alpha$ and $\alpha'$ are the same until $B_\ell$. After that, only $p_i$ can distinguish between the executions, but the next time $p_i$ writes is after all other processes terminate, so they can not distinguish $\alpha$ from $\alpha'$, and $\alpha' \sim_{P=p_i} \alpha$.

The fact that $p_i$ is seen in $\alpha'$ follows easily from the construction: in the first case, $B'_\ell = \{p_i\} \cup B_{\ell+1}$ so $p_i$ is seen by the processes in $B_{\ell+1}$; in the second case, $B'_\ell = \{p_i\}$ and $B'_{\ell+1} = B_\ell \setminus \{p_i\} \neq \emptyset$, so $p_i$ is seen by the processes of $B_\ell \setminus \{p_i\}$.

As to the opposite signs, note that by Proposition 4, a change in the number of steps of the form $\{p_i\}$ at the end of the execution does not affect the sign. All blocks but $B_\ell$ and $B_{\ell+1}$.

Figure 5.1: Proof of Lemma 5—creating a pair of indistinguishable executions (singleton steps by $p_i$ at the ends of the executions do not appear in the figure)
remain unchanged, so their contribution to the sign persists. The only difference between \( \alpha \) and \( \alpha' \) comes from merging \( \{p_i\} \) into \( B_{t+1} \), or splitting it from \( B_t \).

In the first case, \( B_t' = \{p_i\} \cup B_{t+1} \), so

\[
(-1)^{|B_t'|+1} = - \left( (-1)^{|\{p_i\}|+1} \right) \cdot (-1)^{|B_{t+1}|+1},
\]

and in the second case, the opposite transformation is performed. Hence, in both cases, \( \text{sign}(\alpha) = -\text{sign}(\alpha') \), as claimed.

To prove the uniqueness, we use the next claim:

**Claim 6.** Let \( P \) be a set of processes, and let \( p_i \in P \). Let \( \beta \) and \( \beta' \) be two different executions by \( P \) of the same algorithm, induced by the sequences of blocks \( B_1 \cdot \cdot \cdot B_h \) and \( B'_1 \cdot \cdot \cdot B'_h \), respectively. Let \( B_t \) be the first block which is not identical in the sequences, \( B_t \neq B'_t \). If \( \beta \not\sim_{p_i} \beta' \), then \( B_t \cap B'_t = \{p_i\} \).

**Proof of claim.** First, we prove that both \( B_t \) and \( B'_t \) exist, i.e., both executions do not terminate before the \( t \)th block. Assume for contradiction one of the executions terminates in less than \( t \) blocks, and without loss of generality assume this execution is \( \beta \), i.e., \( \beta \) is induced by \( B_1 \cdot \cdot \cdot B_r \), \( r < t \). Both \( \beta \) and \( \beta' \) are executions of the same algorithm, and \( B_j = B'_j \) for every \( j < t \), hence \( \beta' \) is terminates after \( B'_1 \cdot \cdot \cdot B'_r \) as well; so the executions \( \beta \) and \( \beta' \) are identical, which is a contradiction.

Since \( B_t \) and \( B'_t \) are nonempty and different, there is a process \( p_k \in B_t \cup B'_t \), \( p_k \neq p_i \). Assume without loss of generality that \( p_k \in B_t \). \( p_k \) takes a step in \( \beta \) in \( B_t \), so it does not terminate in the partial execution induced by the first \( t - 1 \) blocks, which are identical in both executions; hence, \( p_k \) takes a step after \( B'_{t-1} \) in \( \beta' \) as well. In this step, \( p_k \) takes a scan, and gets a view containing the steps of processes of \( B'_t \); if \( B_t \cap B'_t = \emptyset \), these processes do not appear in the view of \( p_k \) in the scan taken in \( B_t \), so \( p_k \) distinguishes between the executions.

If \( B_t \cap B'_t \neq \{p_i\} \), then there is a process \( p_k \in B_t \cap B'_t \), \( p_k \neq p_i \). In the scan operations corresponding to \( B_t \) and \( B'_t \) in \( \beta \) and in \( \beta' \) respectively, \( p_k \) has views containing the different processes of \( B_t \) and \( B'_t \), hence \( p_k \) distinguishes between the executions. \( \square \)

Assume for contradiction that there is an execution \( \alpha'' \) of the same algorithm as \( \alpha \) and \( \alpha' \), induced by \( B''_1 \cdot \cdot \cdot B''_{h''} \), and satisfying \( \alpha'' \not\sim \alpha \), \( \alpha'' \not\sim \alpha' \), \( \alpha'' \not\sim_{p_i} \alpha \) and \( \alpha'' \not\sim_{p_i} \alpha' \). Let \( B_t \) be the first block satisfying \( B_t \neq B''_t \); by Claim 6, \( B_t \cap B''_t = \{p_i\} \).

Since \( p_i \) sees different processes in the views returned by the scan operations corresponding to \( B_t \) and \( B''_t \), it is in different states after the partial executions induced by \( B_1 \cdot \cdot \cdot B_t \) and by \( B''_1 \cdot \cdot \cdot B''_t \). Specifically, the state of \( p_i \) after the partial executions induced by \( B_1 \cdot \cdot \cdot B_t \) is never reached by \( p_i \) in the execution \( \alpha'' \). For the case \( t < \ell \), recall \( p_i \) is seen in \( B_\ell \), i.e., there is a process \( p_k \neq p_i \) that takes a step in \( \alpha \) in a block \( B_r \), \( r \geq \ell > t \). The view of \( p_k \) returned by
the scan operation corresponding to $B_r$ contains the state of $p_i$ after $B_1 \cdots B_i$, while no scan operation of $p_k$ in $\alpha''$ may contain this state of $p_i$; hence, $p_k$ distinguishes $\alpha$ from $\alpha''$, making the case $t < \ell$ is impossible.

If $t = \ell$ and $B'' \neq B''_i$, i.e., $B_j = B''_j = B''_j^\ell$ for every $j < \ell$, while $B_{\ell}, B'_{\ell}$ and $B''_{\ell}$ are all distinct, assume without loss of generality that $B_{\ell} = \{p_{\ell}\}$ and $B'_{\ell} = \{p_{\ell}\} \cup B'_{\ell+1}$ (i.e., the first case of the construction holds). Since $B_{\ell} \neq B''_{\ell}$, there is a process $p_k \neq p_{\ell}$, satisfying $p_k \in B''_{\ell}$.

By Claim 6, $B'_{\ell} \cap B''_{\ell} = \{p_k\}$, hence $p_k \notin B'_{\ell}$; since $B'_{\ell} = \{p_{\ell}\} \cup B'_{\ell+1}$, we conclude $p_k \notin B'_{\ell+1}$ as well. $p_k$ does not terminate in the first $\ell - 1$ blocks, which are identical for all processes, so it takes a scan in $\alpha$ after $B'_{\ell+1}$ and in $\alpha'$ after $B'_{\ell}$. In these scan operation, $p_k$ has a view containing the steps taken by all processes of $B_{\ell+1}$ in $B_{\ell+1}$ in $\alpha$ and in $B'_{\ell}$ in $\alpha'$; these processes do not take a step in $B''_{\ell}$, so $p_k$ has a different view in its first scan operation after $B_{\ell+1}$, and so it distinguishes $\alpha''$ from $\alpha$ and from $\alpha'$.

If $t = \ell$ and $B'' = B''_i$, denote $B''_j$ the first block satisfying $B'' \neq B''_j$; as $B''_j = B''_j^\ell$ for every $j < \ell$, we conclude $\ell < r$. Recall that $B''_j$ is the last block in which $p_i$ is seen in $\alpha'$; applying Claim 6 to $\alpha'$ and $\alpha''$ implies $p_i \in B''_j$, so all processes but $p_i$ terminate in $\alpha'$ in the partial execution induced by $B''_1 \cdots B''_{r-1}$. But $B''_1 \cdots B''_{r-1}$ is identical to $B''_1 \cdots B''_{r-1}$, so all processes but $p_i$ terminate in the partial execution induced by $B''_1 \cdots B''_{r-1}$ as well. Therefore, $B''_j = \{p_i\} = B''_j$, which is a contradiction to the choice of $r$.

If $t > \ell$, we deduce $B'' = B_t$, and apply analogous arguments. By Claim 6, $p_i \in B_t$; by the choice of $\ell$ as the last block in which $p_i$ is seen and the assumption $t > \ell$, all processes but $p_i$ terminate in $\alpha$ in the partial execution induced by $B_1 \cdots B_{\ell+1}$. Since $B_1 \cdots B_{\ell+1}$ is identical to $B''_1 \cdots B''_{\ell+1}$, all processes but $p_i$ terminate in $\alpha'$ in the partial execution induced by $B_1'' \cdots B''_{\ell+1}$ as well, so $B'' = \{p_i\} = B_t$, contradicting the choice of $B_t$. Therefore, $B_j = B''_j$ for every $j$, $\alpha = \alpha''$, and the uniqueness of $\alpha'$ follows.

The last lemma was also used in Chapter 3, where it was applied to executions by different sets of processes. For the proofs in this chapter, however, we only argue about executions by all processes, hence from now on we consider only such executions.

For an algorithm $A$ and for $v \in \{0, 1\}$, the set of executions of $A$ in which only $v$ is decided is $C_A^v = \{\alpha \mid \alpha \text{ is an execution of } A \text{ by all processes } | \text{dec}(\alpha) = \{v\}\}$. These sets are defined for any algorithm, but for algorithms solving symmetry breaking tasks both sets should be empty, since executions in which all processes decide on the same value are prohibited. For the impossibility proof, we use the next definition.

**Definition 3.** Let $A$ be an algorithm. The univalued signed count of $A$ is defined to be $\sum_{\alpha \in C_A^0} \text{sign}(\alpha) + (-1)^{n-1} \cdot \sum_{\alpha \in C_A^1} \text{sign}(\alpha)$.

Note that if the univalued signed count is nonzero, then at least one of $C_A^0$ and $C_A^1$ is nonempty, hence $A$ has an execution with a single output value. The converse is not necessarily true, but this does not matter for the impossibility result.
5.2 A Trimmed Algorithm

Let $A$ be a wait-free algorithm that produces binary outputs. As explained in Chapter 2, we assume that process $p_i$ alternates between a write operation, a scan operation and a local computation, as follows:

$$
\text{Write}(\text{initialState}_i) \text{ to } R_i
$$

while true do

$\vec{r} \leftarrow \text{Scan} (R_0, \ldots, R_{n-1})$

Local$_A(\vec{r})$: [ computation on $\vec{r}$

if $\text{cond}(\vec{r})$ then return $v(\vec{r})$ ]

Write($\vec{r}$) to $R_i$

We derive from $A$ a trimmed algorithm, $T(A)$. $T(A)$ does not claim to solve any specific task; it is defined such that $A$ and $T(A)$ have the same univalued signed count, and estimating it for $T(A)$ is easier. In $T(A)$, each process conducts a simulation of $A$ as long as it does not see all other processes; if the simulation terminates, $T(A)$ terminates and produces the same output value. Otherwise, $T(A)$ halts when all processes arrive, and outputs 1 if it took simulation steps, or 0 otherwise (see Figure 5.2). The pseudocode of $T(A)$ for a process $p_i$ is:

$\text{simulated} \leftarrow 0$

Write($\text{initialState}_i$) to $R_i$

while true do

$\vec{r} \leftarrow \text{Scan} (R_0, \ldots, R_{n-1})$

if $\vec{r}$ contains all processes then return $\text{simulated}$

$\text{simulated} \leftarrow 1$

Simulate Local$_A(\vec{r})$

if $A$ returns $v$ then return the same value $v$

Write($\vec{r}$) to $R_i$
Since we assume each process writes all its history to the register, and then reads its own
register, we can assume Local\(_A\) does not depend on the state of the process. Hence, we simulate
the steps of a process without explicitly tracking its current state.

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, and every process but \(p_i\) has the
same output in both cases. By Proposition 4, 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 7.** \(A\) and \(T(A)\) have the same univalue 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 2). These tuples
contain executions spanning from the executions where only 0 is decided to the executions
where only 1 is decided: consider tuples of the form \((\alpha, p_i)\) such that in \(\alpha\), all processes with
identifier smaller than \(i\) output 1, and all processes with identifier greater than \(i\) output 0. As
in the proof of Lemma 2, the output of \(p_i\) does not matter. Formally, for an algorithm \(A\), let:

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

Note that every univalued execution appears in \(X^A\) once: if \(\alpha\) is a 0-univalued execution, it
appears only in a tuple \((\alpha, p_0)\); if \(\alpha\) is a 1-univalued execution, it appears only in \((\alpha, p_{n-1})\). Any
other execution that appears in \(X^A\) appears twice, in tuples with processes that have consecutive
identifiers: if \(\alpha\) is an execution where \(\text{dec}(\alpha, \{p_0, \ldots, p_i\}) = \{1\}\) and \(\text{dec}(\alpha, \{p_{i+1}, \ldots, p_{n-1}\}) = \{0\}\) for some \(i\), then it appears in exactly two tuples, \((\alpha, p_i)\) and \((\alpha, p_{i+1})\).

**Definition 4.** Let \(A\) be an algorithm. The signed count of \(A\) is

\[
\sum_{(\alpha, p_i) \in X^A} (-1)^i \text{sign}(\alpha).
\]

The \((-1)^i\) coefficient in the signed count of \(A\) is used to cancel out pairs of tuples consisting
of the same execution and processes with consecutive identifiers (see the proof of Claim 8
below), while the sign ensures that what is left is equal to the univalued signed count, as stated
in the following claim:

**Claim 8.** For any algorithm \(A\), the signed count is equal to the univalued signed count.

**Proof of claim.** Consider two types of tuples \((\alpha, p_i) \in X^A\), according to the output values
decided in \(\alpha\):

1. If \(\alpha \in C^A_0\) then it appears in \(X^A\) once, as \((\alpha, p_0)\). Hence, it is counted in the signed count
   and in the univalued signed count as \(\text{sign}(\alpha)\).

Similarly, if \(\alpha \in C^A_1\) then it appears in \(X^A\) once, as \((\alpha, p_{n-1})\), and counted in the signed
count and in the univalued signed count as \((-1)^{n-1} \text{sign}(\alpha)\).
Figure 5.3: Part of the proof of Lemma 7. A and $T(A)$ have the same signed count.

Tuples with $p_i$ unseen are the same in $X^A$ and in $X^{T(A)}$; tuples with $p_i$ seen cancel out in pairs by Lemma 5.

2. If $\alpha \notin C_0^A \cup C_1^A$, then for some $i$, $0 \leq i < n - 1$,

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

Hence $\alpha$ appears exactly twice in the signed count of $A$, for $(\alpha, p_i)$ and for $(\alpha, p_{i+1})$, and the corresponding summands cancel each other, since $(-1)^i \text{sign}(\alpha) = -(-1)^{i+1} \text{sign}(\alpha)$.

Therefore, every tuple in $X^A$ implies either a summand that appears in the signed count and in the univalued signed count with the same coefficient, or two summands that appear in the signed count with opposite coefficients, and hence cancel each other. On the other hand, every execution $\alpha \in C_0^A$ appears in $X^A$ in a tuple $(\alpha, p_0)$, and every $\alpha \in C_1^A$ appears in $X^A$ in a tuple $(\alpha, p_{n-1})$, as discussed in the first case. Hence, the sums are equal. $\square$

It remains to show that $A$ and $T(A)$ have the same signed count. The proof of this claim is illustrated in Figure 5.3.

For a tuple $(\alpha, p_i) \in X^A$ such that $p_i$ is unseen in $\alpha$, consider the execution $\bar{\alpha}$ of $T(A)$ with the same sequence of blocks, possibly omitting singleton steps at the end; by Proposition 4, both executions have the same sign. Moreover, all processes but $p_i$ complete the simulation of $A$ and output the same values as in $\alpha$. Hence, $(\bar{\alpha}, p_i) \in X^{T(A)}$ and the contribution of $(\alpha, p_i)$ to the signed count of $A$ equals the contribution of $(\bar{\alpha}, p_i)$ to the signed count of $T(A)$.

The sum over tuples $(\alpha, p_i) \in X^A$ in which $p_i$ is seen in $\alpha$ is 0: fix a process $p_i$ and consider all the tuples $(\alpha, p_i) \in X^A$ in which $p_i$ is seen in $\alpha$. By Lemma 5, for each $\alpha$ there is a unique execution $\alpha' \neq \alpha$ of $A$ such that $\alpha \overset{p_i}{\sim} \alpha'$, hence every process other than $p_i$ decides the same
in $\alpha$ and in $\alpha'$, and $(\alpha', p_i) \in X^A$. Moreover, $p_i$ is also seen in $\alpha'$, and $\text{sign}(\alpha) = -\text{sign}(\alpha')$. Hence, we can divide all these tuples into pairs, $(\alpha, p_i)$ and $(\alpha', p_i)$ with $\text{sign}(\alpha) = -\text{sign}(\alpha')$, each of which cancels out in the signed count of $A$. Since Lemma 5 applies to $T(A)$ as well, the same argument shows that the sum over the tuples $(\alpha, p_i) \in X^T(A)$ in which $p_i$ is seen in $\alpha$ is 0 as well.

For an execution $\alpha$ of $T(A)$, denote by $SIM_\alpha$ the set of processes that assign $\text{simulated} = 1$ in $\alpha$, namely processes that take simulation steps of $A$. If $SIM_\alpha = \emptyset$ then $\alpha$ is the unique execution in which all processes take a step together in the first block, hence they all see each other and output 0 without taking further steps; this execution is denoted $\alpha_{\text{all}}$.

For every other execution $\alpha \neq \alpha_{\text{all}}$ of $T(A)$, it holds that $1 \leq |SIM_\alpha| \leq n - 1$: the first process to take a first scan operation sets $\text{simulated} = 1$, while the last process to take a first scan operation has all other processes in its view after its first scan operation, so it outputs 0 and never sets $\text{simulated} = 1$. Hence, $C_1^{T(A)} = \emptyset$.

We conclude the following:

**Proposition 9.** Let $T(A)$ be a trimmed algorithm. Then (i) $\alpha_{\text{all}} \in C_0^{T(A)}$ and (ii) $C_1^{T(A)} = \emptyset$.

### 5.3 Impossibility of SSB and Adaptive Renaming

To prove the impossibility of SSB, consider an SSB algorithm $S$, and its trimmed version, $T(S)$.

**Lemma 10.** If $S$ has the output-one property, then the univalued signed count of $T(S)$ is nonzero.

**Proof.** Consider an execution $\alpha \neq \alpha_{\text{all}}$ of $T(S)$: we show that $\alpha \notin C_0^{T(S)}$. If any of the processes of $SIM_\alpha$ has all other processes in its view while simulating $S$, it decides on the value of its $\text{simulated}$ variable, namely, 1. Otherwise, all processes of $SIM_\alpha$ decide within the simulation of $S$; this is a simulation of a legal execution of $S$ by $SIM_\alpha$, so by the output-one property of $S$, at least one of these processes outputs 1, and $\alpha \notin C_0^{T(S)}$.

Together with Proposition 9(i), this implies that $C_0^{T(S)} = \{\alpha_{\text{all}}\}$; by Proposition 9(ii), $C_1^{T(S)} = \emptyset$, so the univalued signed count of $T(S)$ is $\text{sign}(\alpha_{\text{all}}) = (-1)^{n+1} \neq 0$.

By Lemma 7, the univalued signed count of $S$ equals to the univalued signed count of $T(S)$, so it is also nonzero. Hence, there is an execution of $S$ where all processes output the same value, so the algorithm does not satisfy the symmetry breaking property.

**Theorem 11.** There is no wait-free algorithm solving SSB in an asynchronous shared memory system with any number of processes.

24
If \((2p - \lfloor y/n-1 \rfloor)\)-adaptive renaming is solvable, then so is SSB (see Chapter 4). This implies the next theorem.

**Theorem 12.** There is no wait-free algorithm solving \((2p - \lfloor y/n-1 \rfloor)\)-adaptive renaming in an asynchronous shared memory system with any number of processes.

### 5.4 Impossibility of \(r\)-ISB, WSB and Renaming

Let \(A\) be a symmetric \(n\)-processes \(r\)-ISB algorithm for some \(r\), \(1 \leq r \leq n\), and consider its trimmed version, \(T(A)\). If \(r > 1\), an argument similar to the one used in the proof of Lemma 10 does not apply, as \(A\) does not satisfy the output-one property of an SSB algorithm. Moreover, Lemma 10 is not restricted to symmetric algorithms, whereas, at least for \(r = n\), this restriction is crucial for the impossibility proof: the \(n\)-ISB task is solvable by non-symmetric algorithms, e.g., by letting the process \(p_{n-1}\) decide on 1 and letting any other process decide on 0.

In order to compute the univalued signed count of \(T(A)\), we use the fact that \(A\) is symmetric: we show that every execution \(\alpha\) of \(T(A)\) where some processes take simulation steps has a class of executions with the same outputs as in \(\alpha\); this is formalized by defining an equivalence relation on the executions of \(T(A)\) and considering the equivalence classes it induces. These equivalence classes have predetermined sizes, which depend on \(n\), the number of processes; this allows us to estimate the univalued signed count of \(T(A)\). For some values of \(n\) and \(r\), we show that the univalued signed count of \(T(A)\) cannot be zero; using Lemma 7, we conclude that the univalued signed count of \(A\) is also nonzero, which completes the proof.

Define a relation \(\simeq\) on the executions of \(T(A)\): \(\alpha \simeq \alpha'\) if there is a permutation \(\pi : \{0, \ldots, n - 1\} \to \{0, \ldots, n - 1\}\) that is order preserving on \(SIM_\alpha\) and on \(SIM_{\alpha'}\), such that \(\alpha' = \pi(\alpha)\). Note that \(\simeq\) is an equivalence relation: the identity permutation is order preserving on any set, so \(\simeq\) is reflexive; if \(\pi\) is order preserving on a set \(P\) then \(\pi^{-1}\) is order preserving on \(\pi(P)\), so \(\simeq\) is symmetric; and if \(\pi_1\) is order preserving on a set \(P\) and \(\pi_2\) is order preserving on \(\pi_1(P)\), then \(\pi_2 \circ \pi_1\) is order preserving on \(P\), making \(\simeq\) transitive. The equivalence class of an execution \(\alpha\) is \(\alpha = \{\alpha' \mid \alpha \simeq \alpha'\}\).

Let \(\alpha\) and \(\alpha'\) be two executions of \(T(A)\) satisfying \(\alpha \simeq \alpha'\), and let \(\pi\) be the permutation defining the equivalence. By the definition of \(\simeq\), \(\alpha\) and \(\alpha'\) have the same block structure, and hence the same sign. Since \(A\) is symmetric, if a process \(p_i\) sets \(\text{simulated} = 1\) in \(\alpha\) then \(p_{\pi(i)}\) sets \(\text{simulated} = 1\) in \(\alpha'\). If a process \(p_i\) outputs a value in \(\alpha\) before seeing all other processes, then \(p_{\pi(i)}\) outputs the same value in \(\alpha'\), and in the same number of steps; if \(p_i\) sees all other processes, \(p_{\pi(i)}\) also sees all other processes and, again, \(p_i\) and \(p_{\pi(i)}\) output the same value in the same number of steps. Thus, the equivalence relation \(\simeq\) have the following useful properties:

**Proposition 13.** If \(\alpha \simeq \alpha'\) are two executions of \(T(A)\), then \(\text{sign}(\alpha) = \text{sign}(\alpha')\), \(\vert SIM_\alpha\vert = \vert SIM_{\alpha'}\vert\), and \(\text{dec}(\alpha) = \text{dec}(\alpha')\).
Therefore, we can denote the sign common to 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 \(P'\) to \(P\) and is order preserving on \(P\) and on \(P'\). 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 \(P'\) to \(P\) in such manner. This implies that \(\pi\) is unique, which is used in the proof of the next lemma:

**Lemma 14.** If \(\alpha\) is an execution of \(T(A)\) satisfying \(|SIM_\alpha| = m\), then the size of \([\alpha]\) is \(\binom{n}{m}\).

**Proof.** Let \(\alpha\) be an execution and \(m\) an integer as in the statement of the lemma. Denote by \(\binom{P}{m}\) the set of all subsets of \(P\) of size \(m\); since \(|P| = n\), we have \(\binom{P}{m} = \binom{n}{m}\).

By Proposition 13, if \(\alpha' \in [\alpha]\) then \(SIM_{\alpha'} \in \binom{P}{m}\), so we can define a function \(f: [\alpha] \to \binom{P}{m}\) by \(f(\alpha') = SIM_{\alpha'}\). Next, 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 \(\pi, \varphi\) that are order preserving on \(SIM_\alpha\) and on \(\overline{SIM}_\alpha\). Since \(f(\alpha') = f(\alpha'')\), we have that \(SIM_{\alpha'} = SIM_{\alpha''}\), and hence, \(\overline{SIM}_{\alpha'} = \overline{SIM}_{\alpha''}\). Since there is a unique permutation that maps \(SIM_\alpha\) to \(SIM_{\alpha'}\) and \(\overline{SIM}_\alpha\) to \(\overline{SIM}_{\alpha'}\) and is order preserving on \(SIM_\alpha\) and on \(\overline{SIM}_\alpha\), it follows that \(\pi = \varphi\), thus \(\alpha' = \alpha''\).

Let \(P \in \binom{P}{m}\), and denote by \(\pi\) the unique permutation that maps \(SIM_\alpha\) to \(P\) and \(\overline{SIM}_\alpha\) to \(\overline{P}\) and is order preserving on both sets. Let \(\alpha' = \pi(\alpha)\), so \(|SIM_{\alpha'}| = m\) and \(f(\alpha') = SIM_{\alpha'} = P\). \(\square\)

We can now prove our key lemma.

**Lemma 15.** Let \(q^e\) be a prime power dividing \(n\), and let \(r \leq q^e\). If \(A\) is an \(n\)-processes symmetric algorithm satisfying the \(r\)-output-one property, then the univalued signed count of \(T(A)\) is nonzero.

**Proof.** By Proposition 9(ii), \(C_1^{T(A)} = \emptyset\), implying that the univalued signed count of \(T(A)\) is equal to \(\sum_{\alpha \in C_0^{T(A)}} \text{sign}(\alpha)\). By Proposition 13, if \(\alpha \approx \alpha'\) then \(\text{dec}(\alpha) = \text{dec}(\alpha')\), so for every equivalence class, either \([\alpha] \subseteq C_0^{T(A)}\) or \([\alpha] \cap C_0^{T(A)} = \emptyset\). Hence, \(C_0^{T(A)}\) is the disjoint union of \([\alpha] \subseteq C_0^{T(A)}\), implying

\[
\sum_{\alpha \in C_0^{T(A)}} \text{sign}(\alpha) = \sum_{[\alpha] \subseteq C_0^{T(A)}} \sum_{\alpha' \in [\alpha]} \text{sign}(\alpha').
\]

Recall that all executions in an equivalence class \([\alpha]\) have the same sign, denoted \(\text{sign}([\alpha])\).

By Lemma 14, the size of an equivalence class \([\alpha]\) is \(\binom{n}{|SIM_\alpha|}\). Hence,

\[
\sum_{\alpha \in C_0^{T(A)}} \text{sign}(\alpha) = \sum_{[\alpha] \subseteq C_0^{T(A)}} \left(\binom{n}{|SIM_\alpha|}\right) \text{sign}([\alpha]).
\]

26
We now show that only executions $\alpha$ with $|SIM_{\alpha}| < r$ contribute to the sum. Consider an execution $\alpha$ of $T(A)$ satisfying $|SIM_{\alpha}| \geq r$. If any of the processes of $SIM_{\alpha}$ sees all other processes while simulating $A$, it decides on the value of its simulated variable, namely, 1. Otherwise, all processes of $SIM_{\alpha}$ decide within the simulation of $A$; this is a simulation of a legal execution of $A$ by $SIM_{\alpha}$, so by the $r$-output-one property of $A$, at least one of these processes decides on 1, and $\alpha \notin C_{T(A)}^{0}$. Therefore, the above sum contains only equivalence classes $[\alpha]$ of executions with $0 \leq |SIM_{\alpha}| \leq r - 1$:

$$
\sum_{\alpha \in C_{0}^{T(A)}} \text{sign}(\alpha) = \sum_{m=0}^{r-1} \sum_{|SIM_{\alpha}|=m} \left(\begin{array}{c} n \\ m \end{array}\right) \text{sign}([\alpha]).
$$

We calculate the size of every such equivalence class $[\alpha]$ modulo $q$, using the next result from number theory, which is proved in Chapter 6.

**Claim 16.** If a prime power $q^e$ divides $n$, then $\left(\begin{array}{c} n \\ m \end{array}\right) \equiv 0 \pmod{q}$ for every integer $m$ satisfying $1 \leq m < q^e$.

By Proposition 9, $\alpha_{\text{all}}$ is the unique execution satisfying $|SIM_{\alpha_{\text{all}}}| = 0$. Therefore, all summands except for the one associated with $\alpha_{\text{all}}$ satisfy $1 \leq m < r \leq q^e$, so their contribution to the overall sum is 0 mod $q$. Since $\text{sign}(\alpha_{\text{all}}) = (-1)^{n+1}$, we get

$$
\sum_{\alpha \in C_{0}^{T(A)}} \text{sign}(\alpha) \equiv (-1)^{n+1} \not\equiv 0 \pmod{q},
$$

so $\sum_{\alpha \in C_{0}^{T(A)}} \text{sign}(\alpha) \neq 0$ and the univalued signed count of $T(A)$ is nonzero.

**Theorem 17.** Let $n$ be an integer and let $q^e$ be the largest prime power dividing $n$. If $r \leq q^e$, then there is no symmetric wait-free algorithm solving $r$-ISB in an asynchronous shared memory system consists of $n$ processes.

**Proof.** Let $A$ be a symmetric algorithm for $n$ processes which satisfies the $r$-output-one property. By Lemma 15, the univalued signed count of $T(A)$ is nonzero, and by Lemma 7, the same holds for $A$. This implies that there is an execution of $A$ in which only 0 or only 1 is decided, so $A$ cannot solve $r$-ISB.

As $n$-ISB is the same as WSB, applying this theorem to a prime power $n$ immediately yields the impossibility of WSB with $n$ processes:

**Theorem 18.** There is no symmetric wait-free algorithm solving WSB in an asynchronous shared memory system if the number of processes is a prime power.
If \((2n - 2)\)-nonadaptive renaming is solvable then so is WSB (see Chapter 4). This implies the next theorem.

**Theorem 19.** There is no symmetric wait-free algorithm solving \(n\)-processes \((2n - 2)\)-nonadaptive renaming in an asynchronous shared memory system if \(n\) is a prime power.

For SSB, Theorem 17 implies the unsolvability of SSB in a similar manner, as 1-ISB is SSB. Nevertheless, Theorem 17 holds only for symmetric algorithms, hence, Theorem 11 gives a stronger impossibility result for SSB.

### 5.5 Renaming Lower Bound for Arbitrary Values of \(n\)

In the previous section, we proved that when \(n\) is a prime power, \((2n - 2)\)-nonadaptive renaming is unsolvable. For other values of \(n\), the only known lower bound on the size of the namespace is \(n + 1\) names [6]. This section proves a larger lower bound that holds for all values of \(n\).

Let \(n, M\) be arbitrary positive integers, and assume that there is an algorithm \(A\) solving \(M\)-nonadaptive renaming with \(n\) processes. Since we consider only wait-free algorithms, if \(n' < n\) processes run the algorithm \(A\), they will also solve nonadaptive renaming with range of size \(M\) in a wait-free manner, so \(A\) is also an \(n'\) processes \(M\)-renaming algorithm. If \(n'\) is a prime power and \(M \leq 2n' - 2\), this is impossible by Theorem 19.

On the other hand, Baker et al. [14] show that for a large enough \(n\), there is always a prime number in the interval \([n - n^{0.525}, n]\). This yields the next lower bound for any value of \(n\):

**Proposition 20.** For large enough \(n\), there is no symmetric wait-free algorithm solving \((2n - 2n^{0.525} - 2)\)-nonadaptive renaming in an asynchronous shared memory system.

Cramér [26] conjectured that there is always a primes in a much smaller gap, namely \([n - O(\log^2 n), n]\). If the conjecture is true, the same argument implies:

**Proposition 21.** There is no symmetric wait-free algorithm solving \((2n - \omega(\log^2 n))\)-nonadaptive renaming in an asynchronous shared memory system.
Chapter 6

Divisibility of Binomial Coefficients

Castañeda and Rajsbaum characterize the values of $n$ for which $(2n - 2)$-nonadaptive renaming is unsolvable by the property that $(\binom{n}{1}), \ldots, (\binom{n}{n-1})$ are not relatively prime [20]. In this chapter we explain this characterization is equivalent to $n$ being a prime power.

For an integer $n$ and a prime $q$, let $n = \sum_{j=0}^{s} n_j q^j$ be the base $q$ expansion of $n$, and similarly denote $m = \sum_{j=0}^{s} m_j q^j$. Note that $s$ can be arbitrary large, so we use the same value of $s$ for all the base $q$ expansions in this chapter. Lucas’ Theorem [44] gives an easy way to calculate binomial coefficients modulo $q$:

Lucas’ Theorem

$$\binom{n}{m} \equiv \prod_{j=0}^{s} \binom{n_j}{m_j} \pmod{q}$$

This simple theorem is used to prove the following claims.

Claim 16 If a prime power $q^e$ divides $n$, then $\binom{n}{m} \equiv 0 \pmod{q}$ for every integer $m$ satisfying $1 \leq m < q^e$.

Proof. Assume $q^e$ divides $n$, for $q, e$ as in the statement of the claim, and fix some $m$, $1 \leq m < q^e$. By the choice of $m$, $q^e$ divides $n$ but not $m$, so $q$ divides $\frac{n}{m}$; since $\binom{n}{m} = \frac{n}{m} \binom{n-1}{m-1}$, $q$ also divides $\binom{n}{m}$, as claimed.

The characterization given by Castañeda and Rajsbaum [20] is equivalent to ours, as stated in the next theorem:

Theorem 22. $(\binom{n}{1}), \ldots, (\binom{n}{n-1})$ are not relatively prime if and only if $n$ is a prime power.
The "if" part of this theorem follows easily from the last claim, while the "only if" part was previously noted by Ram (see [28, p. 274]). For the sake of completeness, we give a full proof of this theorem.

**Proof.** Let \( n \) be a power of a prime \( q \), \( n = q^e \). By Claim 16, \( \binom{n}{m} \equiv 0 \pmod{q} \) for every \( 1 \leq m \leq n-1 \), so \( q \) divides all the coefficient \( \binom{n}{1}, \ldots, \binom{n}{n-1} \), hence they are not relatively prime.

For the opposite direction, assume \( n \) is not a prime power, and let \( g = \gcd\left(\binom{n}{1}, \ldots, \binom{n}{n-1}\right) \).

Assume for contradiction \( g > 1 \), and let \( q \) be a prime dividing \( g \). Since \( g \) divides \( \binom{n}{1} = n \), \( q \) also divides \( n \), and let \( e \) be the maximal integer such that \( q^e \) divides \( n \). So \( n = q^et \), for some \( t > 1 \) that \( q \) does not divide.

We show \( q \) does not divide \( \binom{n}{q^e} \).

Since \( q \) does not divide \( t \), \( t \) can be written as \( t = qt' + r \) for some \( 0 < r < q \). So \( n = q^et = q^{e+1}t' + q^er \), hence in \( n \)'s base \( q \) expansion, \( n_e = r \equiv 0 \pmod{q} \). On the other hand, on the base \( q \) expansion of \( q^e \), there is a single nonzero coefficient, 1, on the \( e \)'th place. By Luacs' theorem, \( \binom{n}{q^e} \equiv \prod_{j=0}^{s} \binom{n_j}{q^e} \pmod{q} \), where \( \binom{n_j}{q^e} \) is the \( j \)'th coefficient in \( q^e \)'s base \( q \) expansion. By the facts that there is a single nonzero coefficient in \( q^e \)'s expansion and that \( n_e = r \), it follows that \( \binom{n}{q^e} \equiv \binom{n}{1} = r \equiv 0 \pmod{q} \); hence \( q \) does not divide \( \binom{n}{q^e} \).

On the other hand, \( n \) is not a prime power, so \( 1 \leq q^e \leq n-1 \), and \( q \) does not divide \( \binom{n}{q^e} \), contradicting the fact that \( q \) divides \( g \). Hence, \( g = \gcd\left(\binom{n}{1}, \ldots, \binom{n}{n-1}\right) = 1 \), and the coefficients are relatively prime. \( \square \)
Chapter 7

Conclusions and Open Questions

This thesis presents new proofs of the imposibility of solving distributed decision tasks, using elementary mathematical tools. The first task we consider is \((n - 1)\)-set agreement, which we prove impossible using only an operational lemma and simple parity arguments. Later, we use the SSB task to prove \((2p - \lceil n/p - 1 \rceil)\)-adaptive renaming is not solvable. We define a spectrum of symmetry breaking tasks, spanning from SSB to WSB, and prove these tasks cannot be solved using symmetric algorithms; we then show the impossibility of WSB when \(n\) is a prime power, from which we deduce that \((2n - 2)\)-nonadaptive renaming is not wait-free solvable using symmetric algorithms whenever \(n\) is a prime power. We conclude by using this result and results about the distribution of prime numbers to give a new lower bound for renaming using symmetric algorithms, which holds for any value of \(n\).

This thesis considers only wait-free algorithms, which should be correct even if any number of processes may crash. Nevertheless, it is worth mentioning algorithms for other failure patterns: an algorithm is called \(t\)-resilient if it produces a correct output if at most \(t\) processes fail during its execution. It is known that \(n\)-processes \(t\)-set agreement is solvable using \(t\)-resilient algorithms [17], where \(n > t\); it is also known that if \(n\)-processes \((n - 1)\)-set agreement is not wait-free solvable then \((t - 1)\)-set agreement is not solvable using \(t\)-resilient algorithms [17]. Therefore, from the impossibility of \(n\)-processes \((n - 1)\)-set agreement, impossibility of \(n\)-processes \((t - 1)\)-set agreement with \(t\)-resilient algorithms can also be derived.

7.1 Relation to Previous Proofs and Techniques

The impossibility of \((n - 1)\)-set agreement was previously proved using topological arguments [17,41,49], or graph theoretic arguments [7]. Impossibility of \((2n - 2)\)-nonadaptive renaming for some values of \(n\) was proved only using topological tools [19,21]. The only lower
bound for nonadaptive renaming when \( n \) is not a prime power, namely \( n + 1 \) names are necessary, was proved using operational arguments in the article defining the nonadaptive renaming task [6].

Like prior approaches [7,11,20,41], our proof considers a restricted subset of executions in which all processes output, and in addition, some parts of our proofs are analogous to previous topological proofs [21,41].

In these proofs, a simplicial complex representing all the possible block executions and processes’ views is created. The maximal simplexes of the complex represent executions, and every possible state of each process is represented by a node. The output of a process in a final state is represented by a mapping from this complex to the complex of possible outputs.

Variants of Lemma 1 are used in previous papers to prove that the simplicial complex representing block executions forms a manifold. A face of a simplex representing an execution \( \alpha \) without the node representing \( p_i \) is treated here as the pair \( (\alpha, p_i) \). Indeed, Lemma 2 is analogous to Sperner’s Lemma, and its proof is analogous to a simple proof of Sperner’s Lemma [37]. Attiya and Casteñeda [7] gave a different proof for the impossibility of \((n-1)\)-set agreement, using graph theoretical arguments. Their proof is analogous to another proof of Sperner’s Lemma, which uses the dual graph of a simplicial complex.

The sign of an execution is used here instead of the topological technique of defining an orientation on a manifold and comparing it to the orientation of each simplex; the univalued signed count is the counterpart of the topological notion of content. The trimmed algorithm \( T(W) \) is an operational interpretation of the cone construction [21], with slight modifications. Lemma 7 is achieved in the topological proofs by defining a new coloring of the complex as the local sum of the process identifier and the output bit, and then applying the Index Lemma twice. A topological analog of Proposition 13 is proved in a relatively complicated manner, using \( i \)-corners or flip operations and paths in a subdivided simplex.

Our proofs for the impossibility of \((n-1)\)-set agreement and of adaptive and nonadaptive renaming were inspired by the topological proofs. Nevertheless, they are self contained, much simpler, and more accessible to a reader unfamiliar with topology. Moreover, they give operational counterparts for the topological notions, which we hope will facilitate further investigation of these tasks.

The lower bound for nonadaptive renaming for general values of \( n \) is a new contribution of this thesis; it is achieved by a reduction, and using results about the distribution of primes. The characterization as prime powers is also proved using number theory arguments, namely Lucas’ Theorem about the divisibility of binomial coefficients.
7.2 Future Research

In this thesis we have reproved the impossibility of \((2n - 2)\)-nonadaptive renaming when \(n\) is a prime power. This bound is tight, as \((2n - 1)\)-nonadaptive renaming is wait-free solvable [6]. For other values of \(n\), we show the impossibility of \((2n - n^{0.525} - 2)\)-nonadaptive renaming, while Castañeda and Rajsbaum [22] proved \((2n - 2)\)-nonadaptive renaming is solvable, leaving a large gap between upper and lower bound on the required namespace.

The lower bound for renaming for general values of \(n\) is proved based on gaps between prime numbers, ignoring prime powers with exponent greater than 1. Since prime powers with such exponents are much more sparse than prime numbers (see, e.g., [36]), we believe that including them in the analysis will not yield better bounds, though we were not able to rule out this possibility completely.

We consider a standard model of computation: shared memory, wait-free algorithms, and only block executions. Applying similar techniques to other communication primitives, other failure patterns, or different kinds of executions can be an interesting continuation to this work, and might lead to additional results, or to simpler proofs of known results.

Taking a broader perspective, \(k\)-set agreement is a canonical example of a colorless task, as processes might adopt each other’s input and output values without violating the task’s specification. \((2p - \lceil p/n - 1 \rceil)\)-adaptive renaming and SSB are not colorless tasks in the strict sense, but they are known to be equivalent to \((n - 1)\)-set agreement. \((2n - 2)\)-nonadaptive renaming, on the other hand, is an example of a colored task, as the uniqueness of the outputs prevent processes from adopting each other’s output values. This raises the question of the relation between colored and colorless tasks. While adaptive and nonadaptive renaming seem similar, one is equivalent to a colorless task, while the other is not known to be equivalent to such task; a similar observation holds for SSB and WSB, and this is what motivated the \(r\)-ISB family of tasks.

We hope this thesis will lead to a better understanding of the tasks under consideration. Specifically, nonadaptive renaming have eluded researchers for more than two decades, and we hope the new techniques and lower bounds will help in future investigation of this task. Additionally, the relation between adaptive and nonadaptive renaming is not fully understood, and so is the relation between WSB and SSB. We hope the \(r\)-ISB tasks will help to shed a new light on the relation between these tasks.
Bibliography


[25] S. Chaudhuri. More choices allow more faults: set consensus problems in totally asyn-

[26] H. Cramér. On the order of magnitude of the differences between consecutive prime 

[27] O. Denysyuk and L. Rodrigues. Brief announcement: order-preserving renaming in syn-
chronous message passing systems with byzantine faults. In PODC '12, pages 233–234.

[28] L. Dickson. History of the Theory of Numbers, volume I: Divisibility and Primality of 

[29] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for dis-


[31] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus 


M. Fischer, and M. Yung, editors, Stabilization, Safety, and Security of Distributed Sys-
tems, volume 6366 of Lecture Notes in Computer Science, pages 362–376. Springer Berlin 

[34] E. Gafni, S. Rajbaum, and M. Herlihy. Subconsensus tasks: Renaming is weaker than 
set agreement. In S. Dolev, editor, Distributed Computing, volume 4167 of Lecture Notes 

[35] A. Gottlieb, B. D. Lubachevsky, and L. Rudolph. Basic techniques for the efficient coordi-
nation of very large numbers of cooperating sequential processors. ACM Trans. Program. 

[36] G. Hardy. Ramanujan: Twelve Lectures on Subjects Suggested by His Life and Work. AMS 


In this section, we extend the definitions of certain tasks with parameters and study a specific case where the symmetry is weakly broken but can still be broken with strong symmetry. For tasks of this kind, we prove that there are no solutions using similar methods to those used for the weakly broken symmetry case. We suppose that there exists an algorithm for the task, and we find that it has no solutions. This means that the hypothesis that there exists an algorithm for the broken symmetry case is false, as we have shown that it is impossible to break strong symmetry.

In particular, we consider the case where there is a module that commutes with the task, and we show that the symmetry is broken in this case. We prove that if there is a module that commutes with the task and there is no solution, then the hypothesis that there exists an algorithm for the broken symmetry case is false.

Finally, we summarize the results of this section and list the references for further reading.
.dot

לבצע בכמהLEC🔥החלפה של הכוונה בנוסח השגוי.-motion

זוהי ניסוח של הצורך המתאים לברית סימטריה של הפיתוחים בנושאים שונים.�

לדברי הז'פג, התכנית שלה פרקה על מים מועדים למכירה של התוכן..

החותם של זה הוא קşıי

adroeditor.com
תקציר

chèדוק יתועשר במערכת מבוזרות, הולכת וגוברת, עם התפשטות הרשתות המחשוב והדר נוכחותן של מערכות מבוזרות. המחשבים, טלפונים סלולריים ובמבנים בעלי ביצועים גבוהה ישן קבוצה של מערכות אלה שistenciaן中间ק ומקשתות ביניהן государות.

למערות, האולק המבנה הבסיסי של ליבות מחשבות המתקשרת ביניהן נותר. הגה זה על כל מבנה זה המבנה של מבנה זה על כל מבנה זה המבנה של מבנה זה. מערכות אלו, חליפות נושאות כשתלられている כשבה במעבדה, אינן מבצעות משימות מקצועיות.

הוא עולה בדיקה את יכולות וגבילויות מערכות אלו ובדיקת יכולתן לבצע משימות. בכל תהליך, ערך קלט, ערך פלט, מחשב. התחילינים האחראים, שבם מחולק על ערך פלט מוער.

בהתו וגו בחרואת את המפליאל מלבני והשליפה מקבוצות לועז מערכות התלויות וחופשות מقبلות ניסיון קירור מבית מישורים מצויה מבית רכיבים של מערכות התלויות משימאות מבית.

ככל התלוי עלי תולעי ביצוע האולק

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט מוער, ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.

בכפיפות של הקבוצה הטובה על ערך קלט, ערך פלט, מחשב. התחילין, שבם מחולק על ערך פלט מוער.
המפרס נעשיה בהנחייתה של פרופסור חגי עטיה
בפקולטה למדעי המחשב.

אני מודה לקרן ישראלית לפיד\(\text{\footnotesize 1227/10}\) על תמיכה כספית ונדיבת בдушתיLEEי.

על התרומת הכספית והנדיבת בשיתולים.
לטול
הוכחות אי-אפשרות מבססות מניה
למשימה החלצה מבזורת

יוֹבָר על מתקר

לשם مليוי חלקו של הדרישה ל깥 החלצה הווראה
מניסח למדעי במדעי המחשב

עמרי פ

הוגה last his name - מרכז טכנולוגיה לישראל

2013 מאי

 swingerclub - מחברת החשמל
הוכחות אי-אפטריות מובססות מגייה
למשימות الحلשה מבחרות

עמיע פז