The Universe of Symmetry Breaking Tasks

Damien Imbs, Sergio Rajsbaum, Michel Raynal

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 1965 – janvier 2011

The Universe of Symmetry Breaking Tasks

Damien Imbs[*] , Sergio Rajsbaum[**] , Michel Raynal[***]
damien.imbs@irisa.fr, rajsbaum@math.unam.mx, raynal@irisa.fr

Abstract: Processes in a concurrent system need to coordinate using a shared memory or a message-passing subsystem in order to solve agreement tasks such as, for example, consensus or set agreement. However, often coordination is needed to "break the symmetry" of processes that are initially in the same state, for example, to get exclusive access to a shared resource, to get distinct names or to elect a leader.

This paper introduces and studies the family of *generalized symmetry breaking* (GSB) tasks, that includes election, renaming and many other symmetry breaking tasks. Differently from agreement tasks, a GSB task is "inputless", in the sense that processes do not propose values; the task specifies only the symmetry breaking requirement, independently of the system's initial state (where processes differ only on their identifiers). Among many various characterizing the family of GSB tasks, it is shown that (non adaptive) perfect renaming is universal for all GSB tasks.

Key-words: Agreement, Coordination, Decision task, Election, Disagreement, Distributed computability, Renaming, k-Set agreement, Symmetry Breaking, Universal construction, Wait-freedom.

L'univers des tâches "Symmetry Breaking"

Résumé : *Dans un système réparti, les processus ont besoin de coordination en utilisant un sous-système de mémoire partagée ou de passage de message pour pouvoir résoudre des problèmes tels que le consensus ou l'accord ensembliste. Dans certains cas, la coordination est nécéssaire pour "casser la symétrie" entre des processus qui ont le même état initial.*

Ce rapport introduit la famille des tâches "generalized symmetry breaking" *(GSB) qui inclut l'élection, le renommage et de nombreuses autres tâches qui cassent la symétrie.*

Mots clés : *Accord, Coordination, Tâche de décision, Election, Désaccord, Calculabilité distribuée, Renommage, Accord ensembliste, Symmetry Breaking, Construction universelle, Sans-attente.*

---

[*] Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes
[**] Instituto de Matemáticas, UNAM, Mexico City, Mexico
[***] Membre senior de l'Institut Universitaire de France. Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes

1 Introduction

Processes of a distributed system need to coordinate through a communication medium (shared memory or message-passing subsystem) in order to solve various forms of agreement problems. If no coordination is ever needed in the computation, then we have a set of centralized, independent programs rather than a global distributed computation. Agreement coordination is one of the main issues of distributed computing. As an example, consensus is a very strong form of agreement where processes have to agree on the input of some process. It is a fundamental problem in distributed computing, and the cornerstone when one has to implement a replicated state machine, e.g. [20, 37, 40].

Considering a shared memory asynchronous system where processes may fail by crashing, we are interested here in *tasks* [39], defined by an input/output relation Δ, and where processes start with private input values forming an *input vector* I and, after communication, individually decide on output values forming an *output vector* O, satisfying the specification of the considered task, i.e., $O \in \Delta(I)$. Several specific agreement tasks have been studied in detail, such as consensus [25] and set agreement [21]. Indeed, the importance of agreement is such that it has been studied deeply, from a more general perspective, defining families of agreement tasks, such as loop agreement [33], approximate agreement [23] and convergence [32].

Motivation An important form of coordination is when processes need to disagree. This form of coordination is needed to "break symmetry" among the processes that are initially in the same state. Indeed, specific forms of symmetry breaking have been studied, most notably election, mutual exclusion and renaming. And it is easy to come up with more natural situations related to symmetry breaking. As a simple example, let us consider n persons (processes) such that each one is required to participate in exactly one of m distinct committees (process groups). Each committee has predefined lower and upper bounds on the number of its members. The goal is to design a distributed algorithm that allows these persons (processes) to choose their committees in spite of asynchrony and failures.

Generalized symmetry breaking tasks While the theory of agreement tasks is pretty well developed e.g. [31], it seems that the same substantial research effort has not yet been devoted to understanding symmetry breaking in general. This paper introduces *generalized symmetry breaking* (GSB) tasks, a family of tasks that includes election [41], renaming [7], weak symmetry breaking (called *reduced renaming* in [35]), and many other symmetry breaking tasks. A GSB task for n processes is defined by a set of possible output values, and for each value v, a lower bound and an upper bound (resp., ℓ_v and u_v) on the number of processes that have to decide this value. When these bounds can vary from value to value, we say it is an *asymmetric* GSB task, otherwise we simply say it is a GSB task. For example, we can define the *election* asymmetric GSB task by requiring that exactly one process outputs 1 and exactly $n-1$ processes output 2. In the symmetric case, we use the notation $\langle n, m, \ell, u\rangle$-GSB to denote the task on n processes, for m possible output values, $[1..m]$, where each value has to be decided at least ℓ and at most u times. In the m-renaming task, the processes have to decide new distinct names in the set $[1..m]$. Thus, m-renaming is nothing else than the $\langle n, m, 0, 1\rangle$-GSB task.

Symmetry breaking tasks seem more difficult to study than agreement tasks, because in a symmetry breaking task we need to find a solution given an initial situation that looks essentially the same to all processes. For example, lower bound proofs (and algorithms) for renaming are substantially more complex than for set agreement (e.g., [35]). At the same time, if processes are completely identical, it has been known for a long time that symmetry breaking is impossible [6] (even in failure-free models). Thus, as in previous papers, we assume that processes can be identified by initial names given to them, which are taken from some large space of possible identities (but otherwise they are initially identical). Thus, in an algorithm that solves a GSB task, the outputs of the processes can depend only on their initial identities and on the interleaving of the execution.

When combined with another "output-independence" feature, the symmetry of the initial state of a system differentiates fundamentally GSB tasks from agreement tasks. Namely, the specification of a symmetry breaking task is given simply by a set of legal output vectors, $\mathcal{O}$, that the processes can produce: in any execution, any of these output vectors can be produced for any input vector I (we stress that an input vector defines only the identities of the processes), i.e., $\forall I$ we have $\Delta(I) = \mathcal{O}$. For example, for the election GSB task, $\mathcal{O}$ consists of all binary output vectors with exactly one entry equal to 1 and $n-1$ equal to 2. In contrast, an agreement task typically needs to relate inputs and outputs, where processes should not only agree on closely related values, but in addition the agreed upon values have to be somehow related to the input values given to the processes. Notice that the $\langle n, m, 0, 1\rangle$-GSB renaming task is different from the *adaptive* renaming task, where the size of the new name space depends on the number of processes that participate. Similarly, the classic test-and-set task looks similar to the election GSB task: in both cases exactly one process outputs 1. But test-and-set is adaptive: there is the additional requirement that in every execution, even if less than n processes participate (i.e., take steps), at least one process outputs 1. That is, election GSB is a non-adaptive form of test-and-set.

Contributions This paper investigates the family of GSB tasks in a wait-free setting (where any number of processes can crash). Its main contributions are:

- The introduction of the family of GSB tasks, and a formal setting to study them. It is shown that several tasks that were previously considered separately belong actually to the same family and can consequently be compared and analyzed within a single conceptual framework. Thus, it is shown that several properties that were known for specific GSB tasks, actually hold for all of

them. Moreover, new GSB tasks are introduced that are interesting in themselves, notably the k-*slot* GSB task, the election GSB task and the k-weak symmetry breaking task.

- The structure of the GSB family of tasks is characterized, identifying when two GSB tasks are actually the same task, and giving a unique representation for each one.
- Computability and complexity properties associated with the GSB task family are studied. First it is noticed that (non-adaptive) renaming is a GSB task. It is then shown that perfect renaming (i.e., when the n processes have to rename in the set $[1..n]$) is a universal GSB task. This means that any GSB task can be solved given a solution to perfect renaming. In the other extreme, $(2n-1)$-renaming is trivially solved, without communication. WSB and election are in between these two tasks: they are not solvable without communication. Moreover, election is strictly stronger than weak symmetry breaking.
- As far as the k-slot task is concerned, a simple algorithm is presented that solves the $(n+1)$-renaming task from the $(n-1)$-slot GSB task. There is also a simple algorithm that solves the $(2n-2)$-renaming task from the 2-slot GSB task.

Some of the many interesting questions that remain open are listed in Section 7.

Related work After Dijkstra who mentioned "symmetry" in his pioneering work on mutual exclusion in 1965 [22], the first paper (to our knowledge) to study symmetry in shared memory systems is [13]. It considers two forms of symmetry, and shows that mutual exclusion is solvable only when the weaker form of symmetry is considered. In [41] we encounter for the first time the idea that, although processes have identifiers, there are many more identifiers than processes, and this implies comparison-based algorithms (where the only way to use identities is to compare them). The paper studies the register complexity of solving mutual exclusion and leader election. In contrast, several anonymous models where processes have no identifiers (but where they do have inputs, the opposite of our GSB tasks) have been considered, e.g. [9, 36]. In these models processes do not fail, and yet leader election is not solvable. The papers concentrate then in studying computability and complexity of agreement tasks. In [9] a general form of agreement task function is defined, in which processes have private inputs and processes have to agree on the same output, uniquely defined for each input. A full characterization of the functions that can be computed in this model is presented.

A study comparing the cost of breaking symmetry vs agreement appeared in [24], but again with no failures. It compares the bit complexity cost of agreement vs breaking symmetry in message passing models.

The *weak symmetry breaking* (WSB) task was used in [35] to prove a lower bound on renaming. The task requires processes to decide a binary value, with the restriction that not all decide the same value. Thus, WSB is a GSB task, and its adaptive version, *strong symmetry breaking* (SSB) is not. The SSB task extends this restriction to executions when only a subset of processes participate. It is known that SSB is equivalent to $(n-1)$-set agreement and strictly stronger than WSB [18, 29]. And adaptive $(2n-2)$-renaming can be used to solve $(n-1)$-set agreement [27].

In [26] a family of 01-*tasks* generalizing weak symmetry breaking is defined. As with WSB, never should all processes decide the same binary value. In addition, for executions where not all processes participate, a 01-task specifies a sequence of bits, $b_1, \ldots, b_{n-1}$. If only x processes participate, not all should decide b_i. In contrast, a GSB task specifies restrictions in terms only of n-size vectors (and is not limited to binary values).

An important characteristic of GSB tasks is that their specification does not involve the number of participating processes. This is related to the "output-independence" feature mentioned above, which is not the case with agreement tasks, such as k-test-and-set, k-set agreement, and k-leader election, that are defined in terms of participating sets and, consequently, are adaptive. The three are shown to be related in [14]. In k-*test-and-set* at least one and at most k participating processes output 1. In k-*leader election* a process decides an identifier of a participating process, and at most k distinct identifiers are decided.

Papers considering mixed forms of agreement and symmetry breaking are, group renaming [2, 4], committee decision problem [30] and musical benches [28].

Starting with Angluin [6], covering spaces, more precisely graph coverings, have been used to derive impossibility results in anonymous networks. In these models of distributed computing, processes share a limited knowledge about the underlying communication graph and do not have unique identifiers. For instance, in [19], sufficient and sometimes necessary conditions on the communication graph and the initial, common knowledge are given for fundamental distributed problems such as leader election and enumeration [6, 38, 19]. (Interestingly, [19] is an introduction to local computation in anonymous networks.)

Roadmap The paper is made up of 7 sections. Section 2 presents the computation model. Section 3 defines the GSB tasks. Section 4 investigates the structure of the GSB task family and Section 5 addresses its computability and complexity issues. Section 6 presents a simple algorithm solving $(n+1)$-renaming from the $(n-1)$-slot task. Finally Section 7 lists open challenging problems.

2 Computation model

This paper considers the usual asynchronous, wait-free shared memory system where at most $n-1$ out of n processes can fail by crashing, and the memory is made of single-writer/multi-reader registers. Nevertheless, we restate carefully some aspects of this model

because we are interested in a *comparison-based* and an *index-independent* (called *anonymous* in [7]) solvability notion that are not as common.

2.1 Processes and communication model

Asynchronous crash-prone processes The system includes n asynchronous processes, denoted p_1, ..., p_n. Up to t processes can fail by crashing, $1 \leq t \leq n$ (defined formally below).

Communication objects The processes communicate by reading and writing atomic single-writer/multi-reader (1WnR) registers. Given an array $A[1..n]$ of 1WnR atomic registers, only p_i can write into $A[i]$ while any process can read all entries of A. To simplify the notation in the formal model of this section, we make the following assumptions without loss of generality (they affect efficiency but not computability). The shared memory consists of a single array of 1WnR registers A.[1]. Also, p_i has available a READ operation, such that it gets back a vector of n values, one for each entry of A. The value returned by a READ operation is a snapshot of the array (this assumption is done without loss of generality, because snapshots can be implemented using 1WnR registers, even when $t = n - 1$ [1]). The process p_i also has available a WRITE() operation, such that when p_i invokes it with a parameter val, this value is written to the i-th entry of the register. Finally, the algorithms are *full information*, in the sense that a process always writes its local state (everything it "knows").

Indexes The subscript i (used in p_i) is called the *index* of p_i. Indexes are used only for addressing purposes. Namely, when a process p_i writes a value to A, its index is used to deposit the value in $A[i]$. Also, when p_i reads A, it gets back a vector of n values, where the j-th entry of the vector is associated with p_j. However, we assume that the processes cannot use indexes for computation; we formalize this restriction below.

System model The previous system model is denoted $\mathcal{ASM}_{n,t}[\emptyset]$. The algorithms designed for this computation model have to work despite up to t process crashes. When $1 \leq t \leq n-1$, the model is called the t-resilient model. In the extreme case where $t = n - 1$, the system is called the *wait-free* system model [31].

In Section 5 and Section 6, processes are allowed to cooperate through certain objects, in addition to registers. When the objects implement some task T, the resulting model will be denoted $\mathcal{ASM}_{n,t}[T]$. It is easy to extend the formal model to include these objects.

2.2 Configuration, algorithm and related definitions

Configuration, inputs and outputs A *configuration* of the system consists of the local state of each process and the contents of every atomic register. An *initial configuration* is a configuration in which all processes are in their initial states and each register is given an initial value.

Each process p_i has two specific local variables denoted $input_i$ and $output_i$, respectively. Those are used to solve decision tasks (see below). In an *initial state* of a process p_i, its input is supplied in $input_i$, while its $output_i$ is initialized to a special default value $\perp$. Two initial states of a process differ only in their inputs. Each variable $output_i$ is a write-once variable. A process can write to it only values different from $\perp$, and can write such a value at most once. Hence, as soon as $output_i$ has been written by p_i, its content does not change. A state of p_i with $output_i \neq \perp$ is called an *output state.*

Algorithm, step, run and schedule Each process p_i executes a local algorithm denoted $\mathcal{A}_i$. A *distributed algorithm* is a set $\mathcal{A}$ of n local algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_n$, one per process.

Recall that, in a full information algorithm, a process always writes its local state. A local algorithm consists of a loop: repeatedly writing its state, reading the shared memory, doing local computation and possibly deciding. Thus, all algorithms are identical except for a *decision function* which specifies when a decision is made and what is the decision value. The initial local state of p_i is the value in $input_i$. A process p_i first applies its decision function to possibly modify its local $output_i$ component. Then p_i writes its local state into the shared memory, reads the current value of the shared memory (that becomes its new local state) and repeats this loop.

A *step* (i, Op, w, rs) represents a read (if $Op =$ READ) or a write (if $Op =$WRITE) access to the shared array, by a process p_i. In a write step $rs = \perp$, and p_i issues a write operation with value w, then modifies accordingly its local state. Similarly, in a read step $w = \perp$, and p_i issues a read operation, gets back rs, a vector of values from the shared array, modifies accordingly its local state, possibly writing a decision to $output_i$.

A *run* r is an infinite alternating sequence of configurations and steps $r = C_0\ s_0\ C_1\ \ldots$, where C_0 is an initial configuration and C_{k+1} is the configuration obtained by applying step $s_k = (i, Op, w, r)$ to configuration C_k. The *participating* processes in a run are processes that take at least one step in that run. Those that take a finite number of steps are *faulty* (sometimes called *crashed*), the

[1] Although the codes of our algorithms use more than one register, several registers can be simulated using a single one.

others are *correct* (or *non-faulty*). That is, the correct processes of a run are those that take an infinite number of steps. Moreover, a non-participating process is a faulty process. A participating process can be correct or faulty.

A *schedule* is the sequence of steps of a run, without the values read or written; i.e, only which process took a step and what its operation was. A *view* of process p_i in run r is the sequence of its local states in $C_0\ C_1\ \ldots$ Two runs are *indistinguishable* to a set of processes if all processes in this set have the same view in both runs.

Identities Each process p_i has an identity denoted id_i that is kept in $input_i$. In this paper, we assume identities are the only possible input values. An identity is an integer value in $[1..N]$, where $N > n$ (two identities can be compared with $<$, $=$ and $>$). We assume that in every initial configuration of the system, the identities are distinct: $i \neq j \Rightarrow input_i \neq input_j$.

Clearly, a process "knows" n, because when it issues a read operation, it gets back a vector of n values. However, initially it does not know the identity of the other processes. More precisely, every input configuration where identities are distinct and in $[1..N]$ is possible. Thus, processes "know" N and that no two processes have the same identity.

Index-independent algorithm We say that an algorithm $\mathcal{A}$ is *index-independent* if the following holds, for every run r and every permutation of the process indexes, $\pi()$. Let r_π be the run obtained from r by permuting the input values according to $\pi()$, and for each step, the index i of the process that executes the step is replaced by $\pi(i)$. Then r_π is a run of $\mathcal{A}$.

For example, if in round r process p_1 runs solo with $id_1 = x$, then in r_π we must have that p_2 runs solo with $id_2 = x$, for some $\pi()$. If the algorithm is index-independent, p_2 should behave in r_π exactly as p_1 behaves in r: it decides (writes in $output_i$) the same thing, and in the same step.

Let us observe that in an index-independent algorithm, $output_i = v$ in run r, then $output_{\pi(i)} = v$ in run r_π. This formalizes the fact that indexes are only an addressing mechanism: the output of a process does not depend on indexes, it depends only on the inputs (ids) and on the interleaving. That is, all local algorithms are identical.

Comparison-based algorithm Intuitively, an algorithm $\mathcal{A}$ is *comparison-based* if processes use only comparisons ($<, =, >$) on their inputs.

More formally, let us consider the ordered inputs $i_1 \leq i_2 \leq \cdots \leq i_n$ of a run r of $\mathcal{A}$ and any other ordered inputs $j_1 \leq j_2 \leq \cdots \leq j_n$. The algorithm $\mathcal{A}$ is comparison-based if the run r' obtained by replacing in r each i_ℓ by j_ℓ, $1 \leq \ell \leq n$ (in the corresponding process), is a run of $\mathcal{A}$. Notice that each process decides the same output in both runs, and at the same step.

2.3 Decision tasks

Task A one-shot decision problem is specified by a *task* $(\mathcal{I}, \mathcal{O}, \Delta)$, that consists of a set of *input vectors* $\mathcal{I}$, a set of *output vectors* $\mathcal{O}$, and a relation Δ that associates with each $I \in \mathcal{I}$ at least one $O \in \mathcal{O}$ (e.g. see Section 2.1 of [35]). All vectors are n-dimensional. A task is *bounded* if $\mathcal{I}$ is finite.

Solving a task An algorithm $\mathcal{A}$ *solves* a task T if the following holds: each process p_i starts with an input value (stored in $input_i$) and each non-faulty process eventually decides on an output value by writing it to its write-once register $output_i$. The input vector $I \in \mathcal{I}$ is such that $I[i] = input_i$ and we say "p_i *proposes* $I[i]$" in the considered run. Moreover, the decided vector J is such that (1) $J \in \Delta(I)$, and (2) each p_i decides $J[i] = output_i$. More formally,

Definition 1 *Let $1 \leq t < n$. An n-process algorithm $\mathcal{A}$* solves *a task $(\mathcal{I}, \mathcal{O}, \Delta)$ in $\mathcal{ASM}_{n,t}[\emptyset]$ if the following conditions hold in every run r with input vector $I \in \mathcal{I}$ where at most t processes fail:*

- *Termination. There is a finite prefix of r, denoted $dec\_prefix(r)$, in which for every non-faulty process p_i, $output_i \neq \perp$, in the last configuration of $dec\_prefix(r)$.*
- *Validity. In every extension of $dec\_prefix(r)$ to a run r' where every process p_j ($1 \leq j \leq n$) is non-faulty (executes an infinite number of steps), the values o_j eventually written into $output_j$, are such that $[o_1, \ldots, o_n] \in \Delta(I)$.*

Examples of tasks The most famous task is the *consensus* problem [25]. Each input vector I defines the values proposed by the processes. An output vector is a vector whose all entries contain the same value. Δ is such that $\Delta(I)$ contains all vectors whose single value is a value of I.

The *k-set agreement* task relaxes consensus allowing up to k different values to be decided [21]. Other examples of tasks are *renaming* [7], *weak symmetry breaking* e.g. [35], *committee decision* [30] and *k-simultaneous consensus* [3].

The tasks considered in this paper As already mentioned, this paper considers only tasks where $\mathcal{I}$ consists of all the vectors with distinct entries in the set of integers $[1..N]$. That is, the inputs are the identities. Thus our tasks are bounded. Moreover, we consider only algorithms that are index-independent and comparison-based.

When we consider the system model $\mathcal{ASM}_{n,t}[\emptyset]$ and an algorithm solving a task, for each input vector I, there is an initial configuration whose input values correspond to I. As mentioned before, two processes initially differ only in their identities.

3 The family of generalized symmetry breaking (GSB) tasks

3.1 Definition and basic properties

As already indicated, it is assumed that, in every run, processes start with distinct ids between 1 and N and at most t processes fail. Informally, a generalized symmetry breaking (GSB) task for n processes, $\langle n, m, \vec{\ell}, \vec{u}\rangle$-GSB, $\vec{\ell} = [\ell_1, \ldots, \ell_m]$, $\vec{u} = [u_1, \ldots, u_m]$, is defined by the following requirements. Let us emphasize that the parameters n, m, $\vec{\ell}$ and $\vec{u}$ of a GSB task are statically defined. This means that the GSB tasks are non-adaptive.

- Termination. Each correct process decides a value.
- Validity. A decided value belongs to $[1..m]$.
- Asymmetric agreement. Each value $v \in [1..m]$ is decided by at least ℓ_v and at most u_v processes.

When all lower bounds ℓ_v are equal to some value ℓ, and all upper bounds u_v are equal to some value u, the task is a *symmetric GSB*, and is denoted $\langle n, m, \ell, u\rangle$-GSB, with the corresponding requirement replaced by

- Symmetric agreement. Each value $v \in [1..m]$ is decided by at least ℓ and at most u processes.

To define formally a task, let $\mathcal{I}_N$ be the set of all the n-dimensional vectors with distinct entries in $1, \ldots N$. Moreover, given a vector V, let $\#_x(V)$ denote the number of entries in V that are equal to x.

Definition 2 (GSB Task) *For m, $\vec{\ell}$ and $\vec{u}$, the $\langle n, m, \vec{\ell}, \vec{u}\rangle$-GSB task is the task $(\mathcal{I}_N, \mathcal{O}, \Delta)$, where $\mathcal{O}$ consists of all vectors O such that $\forall\, v \in [1..m] :\ \ell_v \leq \#_v(O) \leq u_v$, and for each $I \in \mathcal{I}_N$, $\Delta(I) = \mathcal{O}$.*

We say that the GSB task is *feasible* if $\mathcal{O}$ is not empty. The following lemma is easy to prove.

Lemma 1 *A GSB task is feasible if and only if $\sum_{v=1}^{m} \ell_v \leq n \leq \sum_{v=1}^{m} u_v$.*

For the case of symmetric GSB tasks, the previous lemma can be re-stated as follows.

Lemma 2 *If $\forall\, v \in [1..m] :\ \ell_v = \ell$ and $\forall\, v \in [1..m] :\ u_v = u$, then the GSB task is feasible if and only if $m \times \ell \leq n \leq m \times u$.*

We fix for this paper $N = 2n - 1$. Thus, all the GSB tasks considered have the same set of input vectors, $\mathcal{I}_{2n-1}$, denoted henceforth simply as $\mathcal{I}$. The following lemma says that considering a set of identities of size larger than $2n - 1$ is useless. A similar result is known for renaming (e.g., [16]).

Theorem 1 *Consider two $\langle n, m, \vec{\ell}, \vec{u}\rangle$-GSB tasks, $(\mathcal{I}_N, \mathcal{O}, \Delta)$, $N \geq 2n - 1$, and $(\mathcal{I}, \mathcal{O}, \Delta)$ (whose only difference is in the set of input vectors). Then $(\mathcal{I}_N, \mathcal{O}, \Delta)$ is wait-free solvable if and only if $(\mathcal{I}, \mathcal{O}, \Delta)$ is wait-free solvable.*

Proof If $(\mathcal{I}_N, \mathcal{O}, \Delta)$ is wait-free solvable so is $(\mathcal{I}, \mathcal{O}, \Delta)$, because $\mathcal{I}$ is a subset of $\mathcal{I}_N$.

Assume that there is a wait-free algorithm $\mathcal{A}$ that solves $(\mathcal{I}, \mathcal{O}, \Delta)$. To solve $(\mathcal{I}_N, \mathcal{O}, \Delta)$, processes get new intermediate identities using any index-independent $(2n - 1)$-renaming algorithm, such as the one in [11], running it with their initial identities from $\mathcal{I}_N$. The intermediate identities obtained belong to $\mathcal{I}_{2n-1} = \mathcal{I}$. The processes run $\mathcal{A}$ using these identities, to solve $(\mathcal{I}, \mathcal{O}, \Delta)$. The outputs produced by this algorithm belong to $\mathcal{O}$, and a solution to $(\mathcal{I}_N, \mathcal{O}, \Delta)$ is obtained. $\square_{Theorem\ 1}$

Recall that an algorithm is comparison-based if processes use only comparison operations on their inputs. The following lemma generalizes another known (e.g., [16, 18]) property about renaming and weak symmetry breaking. It states that we can assume without loss of generality that a GSB algorithm is comparison-based. This is useful to prove impossibility results (e.g., [10, 17]).

Theorem 2 *Consider an $\langle n, m, \vec{\ell}, \vec{u}\rangle$-GSB task, $T = (\mathcal{I}, \mathcal{O}, \Delta)$. There exists a wait-free algorithm for T if and only if there exist a comparison-based wait-free algorithm for T.*

Proof Assume there is a wait-free algorithm $\mathcal{A}$ for T. To get a comparison-based wait-free algorithm for T, first processes obtain new, temporary identities invoking any comparison-based $(2n - 1)$-renaming algorithm, such as the one in [11], running it with their initial identities from $\mathcal{I}$. The intermediate identities obtained belong again to $\mathcal{I}_{2n-1} = \mathcal{I}$. But now the processes use these identities to run $\mathcal{A}$, and solve T, and the resulting algorithm is comparison-based. The other direction holds trivially. $\square_{Theorem\ 2}$

3.2 Instances of generalized symmetry breaking tasks

Let us remember that the parameters n, m, $\vec{\ell}$ and $\vec{u}$ that define a GSB task are statically defined.

Election We can define the *election* asymmetric GSB task, by requiring that exactly one process outputs 1 and exactly $n-1$ processes output 2.

While election is a GSB task with asymmetric agreement, in this paper, we consider mostly GSB tasks with symmetric agreement. This means that the m values are *equal* with respect to decision. If, in a correct run r, v is decided by x processes and w is decided by y processes, then the run r' in which v is decided y processes, w is decided x by processes and the other values are decided as in r, is a correct run. The following are examples of symmetric GSB tasks.

k-Weak symmetry breaking with $k \leq n/2$ (k-WSB) This is the $\langle n, 2, k, n-k\rangle$-GSB task which has a pretty simple formulation. A process has to decide one of two possible values, and each value is decided by at least k and at most $(n-k)$ processes. Let us notice that 1-WSB is the well-known weak symmetry breaking (WSB) task.

m-Renaming In the m-renaming task the processes have to decide new distinct names in the set $[1..m]$. It is easy to see that m-renaming is nothing else than the $\langle n, m, 0, 1\rangle$-GSB task.[2]

Perfect renaming The *perfect renaming* task is the renaming task instance whose size m of the new name space is "optimal" in the sense that there is no solution with $m' < m$ whatever the system model. This means that $m = n$. It is easy to see that this is the $\langle n, n, 1, 1\rangle$-GSB task.

k-Slot This is a new task, defined as follows. Each process has to decide a value in $[1..k]$ and each value has to be decided at least once. This is the $\langle n, k, 1, n\rangle$-GSB task, or its synonym, the $\langle n, k, 1, n-k+1\rangle$-GSB task. As we can see the WSB task is nothing else than the 2-slot task.

We will study in Section 5 the difficulty of solving GSB tasks, and their relative power, and we will discuss the difficulty of each one of the previous GSB tasks. As we shall see, some GSB tasks are solved trivially (i.e., with no communication at all). As an example, this is the case of m-renaming, $m = 2n-1$, namely the $\langle n, 2n-1, 0, 1\rangle$-GSB task (as processes have identities between 1 and $2n-1$, a process can directly decide its own identity). In contrast, some GSB tasks are not wait-free solvable, such as perfect renaming. In fact, we shall see that perfect renaming is universal among GSB tasks.

Tasks that are not GSB tasks *Colorless* tasks are decision tasks that do not care about which process has which input and which process has which output. More precisely, constrained by the relation Δ, they are such that any input legal for one process is legal for the others, and the same is true for their outputs. Consensus and k-set agreement are the most popular colorless tasks. Colorless tasks have been well studied (e.g., [12, 34, 35]). It is easy to see that colorless tasks are not GSB tasks. For example, in a colorless task, if an input vector containing some value v belongs to the task, then the input vector that has all entries equal to v also belongs to the task, while in a GSB task an input vector never has two entries equal to the same value.

4 The structure of symmetric GSB tasks

This section studies the combinatorial structure of symmetric GSB tasks, to analyze the following two issues: synonyms and containment of output vectors. Complexity issues are addressed in Section 5.

Notice that $G_1 = \langle n, m, \vec{\ell}_1, \vec{u}_1\rangle$-GSB and $G_2 = \langle n, m, \vec{\ell}_2, \vec{u}_2\rangle$-GSB may actually be the same task T (i.e., both have the same set of output vectors). In this case we write $G_1 \equiv G_2$, and say that G_1 and G_2 are *synonyms*. For example, $\langle n, 2, 1, n-1\rangle$-GSB, $\langle n, 2, 0, n-1\rangle$-GSB, and $\langle n, 2, 1, n\rangle$-GSB are synonyms.

Also, if the set $S(T_1)$ of the outputs vectors of a GSB task T_1 is contained in the set $S(T_2)$ of the outputs vectors of a GSB task T_2, then clearly T_2 cannot be more difficult to solve than T_1. As $S(T_1) \subset S(T_2)$, any algorithm solving T_1 also solves T_2. In this case, we write $T_1 \subset T_2$.

[2] If m depends on the number of participating processes, the problem is called *adaptive* m-renaming task which is not a GSB task.

4.1 Counting vectors and kernel vectors associated with a task

Let T be an $\langle n, m, \ell, u\rangle$-GSB task defined by the set of output vectors $S(T)$. We associate with T a set of vectors (called *counting vectors* and *kernel vectors*) defined as follows.

Definition 3 *Let $O \in S(T)$. The* counting vector V *associated with O is the m-dimensional vector such that $\forall\ v \in [1..m]$: $V[v] = \#_v(O)$. Let $C(T)$ be the set of counting vectors associated with T.*

It follows from the fact that we consider symmetric agreement, that the counting vectors containing the very same values (e.g., $[a, b, c]$, $[b, c, a]$ and $[c, a, b]$ when considering $m = 3$) can be represented by a single counting vector $K[1..m]$, namely, the single vector whose each entry is greater or equal to the next one (e.g., the counting vector $[b, c, a]$ if $b \geq c \geq a$). Such a vector represents all the output vectors of $S(T)$ in which the most frequent value appears $K[1]$ times, the second most frequent value appears $K[2]$ times, etc.

Definition 4 *Let us partition $C(T)$ into sets X of counting vectors such that each set X contains all the counting vectors that are permutation of each other.*

- *The* kernel vector *of X is its counting vector K such that $K[1] \geq K[2] \geq \cdots \geq K[m]$.*
- *The* kernel set *of T is the set of all its kernel vectors.*
- *The* balanced kernel vector *of T is its kernel vector such that $[\frac{n}{m}, \cdots, \frac{n}{m}]$ if n is a multiple of m, and $K = [\lceil \frac{n}{m} \rceil, \cdots, \lfloor \frac{n}{m} \rfloor]$ (with the first $n \mod m$ entries equal to $\lceil \frac{n}{m} \rceil$) if n is not a multiple of m.*

The next lemma follows directly from the definition of *kernel vector* and *kernel set*.

Lemma 3 *Given a task T, its kernel set is totally ordered by the (usual) lexicographical ordering.*

Summarizing,

- The set of $\langle n, -, -, -\rangle$ GSB tasks is partially ordered (according to the inclusion relation on kernel sets defining tasks),
- If $T1 \subset T2$, any vector (solution) of $T1$ is a vector (solution) of $T2$ from which we conclude that any algorithm that solves $T1$ solves also $T2$.

Examples All the $\langle n, m, \ell, u\rangle$-GSB tasks that are feasible with $n = 6$, $m = 3$ and $u \leq n = 6$ are described in Table 1. Hence, the 6 processes can decide up to 3 different values. The kernel vectors of each of these tasks is indicated, and these kernel vectors are listed according to their lexicographical order, from left to right.

As an example, the kernel vector $[4, 2, 0]$ represents all the output vectors in which the most frequent value (that is 1, 2 or 3) appears 4 times, the second most frequent value appears twice and the third possible value does not appear. As another example, the kernel set of the $\langle 6, 3, 0, 4\rangle$-GSB task is made up of five kernel vectors, namely, $\{[4, 2, 0], [4, 1, 1], [3, 3, 0], [3, 2, 1], [2, 2, 2]\}$. Let us finally observe that the balanced kernel vector $[2, 2, 2]$ belongs to all tasks. Moreover, the GSB tasks $\langle 6, 3, 2, 5\rangle$, $\langle 6, 3, 2, 4\rangle$, $\langle 6, 3, 2, 3\rangle$, $\langle 6, 3, 0, 2\rangle$, $\langle 6, 3, 1, 2\rangle$ and $\langle 6, 3, 2, 2\rangle$ are synonyms. Also, the GSB tasks $\langle 6, 3, 1, 6\rangle$, $\langle 6, 3, 1, 5\rangle$ and $\langle 6, 3, 1, 4\rangle$ are synonyms. Differently, while some tasks are "included" in other tasks (e.g., the kernel vectors associated with any task are included in the kernel set of the $\langle 6, 3, 0, 6\rangle$-GSB task, there are tasks that are not included one in the other (e.g., the $\langle 6, 3, 1, 4\rangle$-GSB and $\langle 6, 3, 0, 3\rangle$-GSB tasks).

Remark It is important to notice that, while a set of kernel vectors can be associated with a task, any set of kernel vectors does not define a task. As an example, a simple look at Table 1 shows that the set of kernel vectors $\{[5, 1, 0], [4, 2, 1]\}$ does not define a task.

4.2 The classes of ℓ-anchored, u-anchored and (ℓ, u)-anchored tasks

This section presents subclasses of GSB tasks that provide us with a better insight on their family structure. More precisely, when we look at the tasks described in Table 1, we see that several GSB tasks are actually synonyms. Hence, it is important to have a single representative for all the GSB tasks that define the same task. This is captured by the notions of ℓ-anchored u-anchored tasks.

Definition 5 (Anchoring) *Let G be an $\langle n, m, \ell, u\rangle$-GSB task, G' be the $\langle n, m, \ell, \min(n, u+1)\rangle$-GSB task and G'' be the $\langle n, m, \max(0, \ell-1), u\rangle$-GSB task. G is* ℓ-anchored *if G and G' are synonyms. G is* u-anchored *if G and G'' are synonyms. G is* (ℓ, u)-anchored *if it is both ℓ-anchored and u-anchored.*

Hence, if G is ℓ-anchored, increasing the upper bound u does not modify the task and, if G is u-anchored, decreasing the lower bound ℓ does not modify the task. Finally, (as we will see) an (ℓ, u)-anchored $\langle n, m, \ell, u\rangle$-GSB task is the hardest of the family of $\langle n, m, -, -\rangle$ GSB tasks.

As an example let us consider the family of $\langle 20, 4, -, -\rangle$-GSB tasks. The reader can easily check that $\langle 20, 4, 4, 8\rangle$ is an ℓ-anchored task, $\langle 20, 4, 2, 6\rangle$ is a u-anchored task, $\langle 20, 4, 5, 5\rangle$ is an (ℓ, u)-anchored task while $\langle 20, 4, 4, 6\rangle$ is neither an ℓ nor a u-anchored task.

It is easy to see that all $\langle n, m, \ell, n\rangle$ (resp., $\langle n, m, 0, u\rangle$) GSB tasks are ℓ-anchored (resp., u-anchored). These tasks are said to be *trivially* anchored.

kernel vector → task ↓	canonical 4-uple	$[6,0,0]$	$[5,1,0]$	$[4,2,0]$	$[4,1,1]$	$[3,3,0]$	$[3,2,1]$	$[2,2,2]$
$\langle 6,3,0,6\rangle$	yes	x	x	x	x	x	x	x
$\langle 6,3,1,6\rangle$					x		x	x
$\langle 6,3,0,5\rangle$	yes		x	x	x	x	x	x
$\langle 6,3,1,5\rangle$					x		x	x
$\langle 6,3,2,5\rangle$								x
$\langle 6,3,0,4\rangle$	yes			x	x	x	x	x
$\langle 6,3,1,4\rangle$	yes				x		x	x
$\langle 6,3,2,4\rangle$								x
$\langle 6,3,0,3\rangle$	yes					x	x	x
$\langle 6,3,1,3\rangle$	yes						x	x
$\langle 6,3,2,3\rangle$								x
$\langle 6,3,0,2\rangle$								x
$\langle 6,3,1,2\rangle$								x
$\langle 6,3,2,2\rangle$	yes							x

Table 1: Kernels of $\langle n,m,\ell,u\rangle$-GSB tasks (with $n=6$ and $m=3$)

Canonical representative of a GSB task Given an $\langle n,m,\ell,u\rangle$-GSB ℓ-anchored task, its *canonical representative* is the $\langle n,m,\ell,u'\rangle$-GSB task such that the $\langle n,m,\ell,u'-1\rangle$-GSB task is not ℓ-anchored. A similar definition applies for an u-anchored task. A task that is neither only ℓ-anchored nor only u-anchored, or that is (ℓ,u)-anchored, is its own representative.

As an example, let us look at Table 1. The $\langle 6,3,2,2\rangle$-GSB task, that is (ℓ,u)-anchored task, is the representative for four tasks associated with the single kernel vector $[2,2,2]$. The $\langle 6,3,1,4\rangle$-GSB task, that is ℓ-anchored, is the representative for three tasks associated with the kernel set $\{[4,1,1],[4,1,1],[2,2,2]\}$. Finally, the $\langle 6,3,1,3\rangle$-GSB task, that is not anchored, is its own representative: it is the only task associated with the kernel set $\{[3,2,1],[2,2,2]\}$.

When considering Table 1 there are 7 canonical representative tasks. These canonical tasks are represented in Figure 1 where "$A \to B$" means "A strictly includes B". Let us notice that the representative $\langle 6,3,1,3\rangle$-GSB task is not anchored.

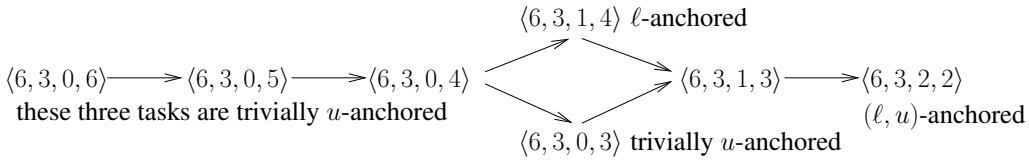

Figure 1: Canonical $\langle n,m,-,-\rangle$ GSB tasks are partially ordered

4.3 A characterization of ℓ-anchored and u-anchored GSB tasks

Let us remember that a task is *feasible* if its set of output vectors $\mathcal{O}$ is not empty.

Theorem 3 *Let T be the feasible $\langle n,m,\ell,u\rangle$-GSB task. T is ℓ-anchored if and only if $u \geq n-\ell(m-1)$.*

Proof Let us first suppose that $n-\ell(m-1) > u \geq \ell$. As $n-\ell(m-1) \geq u+1$, there is a vector (with m entries) whose first entry is equal to $u+1$ that is a kernel vector of the $\langle n,m,\ell,u+1\rangle$ GSB task. But, as $u+1>u$, this vector cannot be a kernel vector of the $\langle n,m,\ell,u\rangle$ GSB task. It follows that the $\langle n,m,\ell,u\rangle$ GSB task cannot be ℓ-anchored.

Let us now suppose that $u \geq n-\ell(m-1) \geq \ell$ and consider the counting vector $[n-\ell(m-1),\ell,\ldots,\ell]$ (with m entries). The sum of all its entries is n. Because the occurrence number $n-\ell(m-1)$ is the only value higher than ℓ, it is the highest value that can appear in a kernel vector of both the $\langle n,m,\ell,u\rangle$ task and the $\langle n,m,\ell,u+1\rangle$ for all $u \geq n-\ell(m-1)$. It follows that the $\langle n,m,\ell,u\rangle$ and $\langle n,m,\ell,u+1\rangle$ GSB tasks are the same GSB task from which we conclude that $\langle n,m,\ell,u\rangle$ is ℓ-anchored. $\square_{Theorem\ 3}$

Theorem 4 *Let T be a feasible $\langle n,m,\ell,u\rangle$-GSB task. T is u-anchored if and only if $\ell \leq n-u(m-1)$.*

Proof The reasoning is similar to the one of Theorem 3. $\square_{Theorem\ 4}$

The next corollary follows from the previous theorems.

Corollary 1 *Let $\ell \leq \frac{n}{m} \leq u$. The $\langle n,m,\ell,\max(\ell,n-\ell(m-1))\rangle$-GSB task is ℓ-anchored, while the $\langle n,m,\max(0,n-u(m-1)),u\rangle$-GSB task is u-anchored.*

4.4 The structural results

Lemma 4 *Let T be any $\langle n, m, \ell, u\rangle$-GSB task. Let $u' \geq u$ and T' be the $\langle n, m, \ell, u'\rangle$-GSB task. We have $S(T) \subseteq S(T')$.*

Proof The only difference between T and T' is the upper bound on the number of processes that can decide the same value. If at most u processes decide each value, then necessarily less than u' processes decide each value, and thus each output vector of the $\langle n, m, \ell, u\rangle$ GSB task T is also an output vector of the $\langle n, m, \ell, u'\rangle$ task T' and consequently e $S(T) \subseteq S(T')$. $\square_{Lemma\ 4}$

Lemma 5 *Let T be any $\langle n, m, \ell, u\rangle$-GSB task. Let $\ell' \leq \ell$ and T' be the $\langle n, m, \ell', u\rangle$-GSB task. We have $S(T) \subseteq S(T')$.*

Proof The reasoning is similar to the one of Lemma 4. $\square_{Lemma\ 5}$

The next theorem characterizes the hardest task of the sub-family of $\langle n, m, -, -\rangle$-GSB tasks. Let us remember that T_1 is harder than T_2 if $S(T_1) \subset S(T_2)$.

Theorem 5 *The $\langle n, m, \lfloor \frac{n}{m} \rfloor, \lceil \frac{n}{m} \rceil\rangle$-GSB task T is the hardest task of the family of feasible $\langle n, m, -, -\rangle$-GSB tasks.*

Proof As we consider only feasible tasks, we have $\ell \leq \frac{n}{m} \leq u$. The proof follows then directly from Lemma 4 and Lemma 5. $\square_{Theorem\ 5}$

Let us observe that, given n and m, the $\langle n, m, \lfloor \frac{n}{m} \rfloor, \lceil \frac{n}{m} \rceil\rangle$-GSB task is not necessarily an anchored task. As an example, the $\langle 10, 4, 2, 3\rangle$-GSB task is neither ℓ-anchored nor u-anchored while the $\langle 10, 5, 2, 2\rangle$-GSB task is (ℓ, u)-anchored.

Theorem 6 *Let T be a feasible $\langle n, m, \ell, u\rangle$-GSB task, $T1$ be the $\langle n, m, \ell', u\rangle$-GSB task where $\ell' = n - u(m-1)$ and $T2$ be the $\langle n, m, \ell, u'\rangle$-GSB task where $u' = n - \ell(m-1)$. We have the following: (i) $(\ell' \geq \ell) \Rightarrow S(T1) \subseteq S(T)$ and (ii) $(u' \leq u) \Rightarrow S(T2) \subseteq S(T)$.*

Proof We prove the theorem for case (i). (The proof for case (ii) is similar.) Let us first show that the $\langle n, m, \ell', u\rangle$-GSB task is feasible, i.e., $\ell' \leq \frac{n}{m} \leq u$. Let us first observe that, as the $\langle n, m, \ell, u\rangle$-GSB task is feasible, by assumption we have $\frac{n}{m} \leq u$. Hence we have only to show that $\ell' \leq \frac{n}{m}$ which is obtained from the following (remember that $m > 1$):

$$\begin{array}{rcrcl} & & n/m \leq & u & \Leftrightarrow \quad n \leq u \cdot m \\ \Leftrightarrow & & n(m-1) \leq & u \cdot m(m-1) & \Leftrightarrow \quad n \cdot m - u \cdot m^2 + u \cdot m \leq n \\ \Leftrightarrow & & \ell' = n - u \cdot m + u \leq & n/m. & \end{array}$$

As $\ell' = n - u(m-1) \leq \frac{n}{m} \leq u$, the size m vector $[u, \ldots, u, \ell']$ is a kernel vector of the feasible $\langle n, m, \ell', u\rangle$ GSB task. As $\ell' \geq \ell$, this vector is also a kernel vector of $\langle n, m, \ell, u\rangle$ GSB task, which concludes the proof for case (i). $\square_{Theorem\ 6}$

The theorem that follows identifies the canonical representative of any feasible $\langle n, m, \ell, u\rangle$-GSB task.

Theorem 7 *Let T be a feasible $\langle n, m, \ell, u\rangle$-GSB task and $f()$ be the function $f(\ell, u) = (\ell', u')$ where $\ell' = \max(\ell, n - u(m-1))$ and $u' = \min(u, n - \ell(m-1))$. The canonical representative of T is the $\langle n, m, \ell_{fp}, u_{fp}\rangle$-GSB task T_{fp} where the pair (ℓ_{fp}, u_{fp}) is the fixed point of $f(\ell, u)$.*

Proof Let us first observe that, using the same reasoning as in Theorem 6, we have $\ell' \leq \frac{n}{m} \leq u'$, from which follows that T_{fp} is feasible (Lemma 2). Moreover, due to the definition of ℓ' and u', we also have $0 \leq \ell \leq \ell' \leq \frac{n}{m} \leq u' \leq u \leq n$. We consider four cases.

- Case $\ell \geq n - u(m-1)$ and $u \leq n - \ell(m-1)$. We have then trivially $\ell' = \ell$ and $u' = u$, from which we conclude that $S(T)$ and $S(T_{fp})$ have the same kernel vectors.
- Case $\ell' = n - u(m-1) > \ell$ and $u' = u$. Let us consider the kernel vector of T that has as many entries as possible equal to $u = u'$. This means that this vector has $m - 1$ entries equal to $u = u'$, and its last entry is equal to $n - u'(m-1)$, i.e., equal to ℓ'. It follows that $S(T)$ has no kernel vector with an entry equal to $\ell'' < \ell'$. We conclude from that observation that the kernel vectors of T are also kernel vectors of T_{fp}, i.e., $S(T) = S(T_{fp})$.
- Case $\ell' = \ell$ and $u' = n - \ell(m-1) < u$. This case is similar to the previous one. Let us consider the kernel vector of T that has as many entries as possible equal to $\ell = \ell'$. This means that this vector has $m - 1$ entries equal to $\ell = \ell'$, and its last entry is equal to $n - \ell'(m-1)$, i.e., equal to u'. It follows that $S(T)$ has no kernel vector with an entry equal to $u'' > u'$. Hence, the kernel vectors of T are also kernel vectors of T_{fp}, i.e., $S(T) = S(T_{fp})$.

- Case $\ell' = n - u(m-1) > \ell$ and $u' = n - \ell(m-1) < u$. This case is a simple combination of both previous cases (one addresses the kernel vectors of T with the greatest possible entries, and the other addresses the kernel vectors of T with the smallest possible entries).

According to Theorems 3 and 4, neither the $\langle n, m, \ell'', u\rangle$-GSB task with $\ell'' > \ell'$ nor the $\langle n, m, \ell, u''\rangle$-GSB task with $u'' < u'$ are synonyms of T, which concludes the proof of the Theorem. $\square_{Theorem\ 7}$

5 Complexity and computability

Recall that for an $\langle n, m, \vec{\ell}, \vec{u}\rangle$-GSB task $T = (\mathcal{I}, \mathcal{O}, \Delta)$, we have that $\Delta(I) = \Delta(I') = \mathcal{O}$, for any two input vectors I, I'. Thus, at first sight, it could seem that a trivial solution for T could be to simply pick a predefined output vector $O \in \mathcal{O}$, and always decide it without any communication, whatever the input vector. This is not the case, in fact, there are GSB tasks that are not wait-free solvable (with *any* amount of communication).

This section investigates the difficulty of solving GSB tasks. In particular, it considers wait-free solvable GSB tasks, i.e., for which there exists an algorithm in the model $\mathcal{ASM}_{n,n-1}[\emptyset]$. The following definition is used to study their relative power.

Definition 6 *A task $T1$ is* stronger than *a task $T2$ (denoted $T1 \succ T2$) if there is an algorithm that solves $T2$ in $\mathcal{ASM}_{n,n-1}[T1]$ ($\mathcal{ASM}_{n,n-1}[\emptyset]$ enriched with an object solving $T1$).*

As we shall see, the universe of GSB tasks includes trivial tasks that can be solved without accessing the shared memory, and universal tasks, that can be used to solve any other GSB task. And in between, there are wait-free solvable tasks, as well as non-wait-free solvable tasks.

5.1 Hardest GSB tasks: Universality of the $\langle n, n, 1, 1\rangle$-GSB task

When considering the GSB family of tasks, an interesting question is the following: is there a universal GSB task? In other words, is there a GSB task that allows other GSB task on n processes to be solved? The answer is "yes". We show in the following that the perfect renaming $\langle n, n, 1, 1\rangle$-GSB task allows any task of the family to be solved. Hence, perfect renaming is *universal* for the family of $\langle n, -, -, -\rangle$-GSB tasks.

As we will see with Corollary 5, the $\langle n, n, 1, 1\rangle$-GSB task (perfect renaming) is not a wait-free solvable task.

Theorem 8 *Any $\langle n, m, \vec{\ell}, \vec{u}\rangle$-GSB task can be solved from any solution to the $\langle n, n, 1, 1\rangle$-GSB task.*

Proof Let us first observe that the $\langle n, n, 1, 1\rangle$-GSB task has a single kernel vector, namely, $[1, \ldots, 1]$. Given an algorithm solving that task, let dec_i be the output at process p_i.

To solve the symmetric $\langle n, m, \ell, u\rangle$-GSB task, the processes execute an algorithm solving the $\langle n, n, 1, 1\rangle$-GSB task, and a process p_i considers $output_i = ((dec_i - 1) \mod m) + 1$ as its output. The corresponding kernel vector for m output values is then $[\lceil \frac{m}{n} \rceil, \ldots, \lceil \frac{m}{n} \rceil, \lfloor \frac{m}{n} \rfloor, \ldots, \lfloor \frac{m}{n} \rfloor]$. By the feasibility assumption, we have $\ell \leq \frac{m}{n} \leq u$. As ℓ and u are integers, we have $\ell \leq \lfloor \frac{m}{n} \rfloor \leq \lceil \frac{m}{n} \rceil \leq u$. The vector $[\lceil \frac{m}{n} \rceil, \ldots, \lceil \frac{m}{n} \rceil, \lfloor \frac{m}{n} \rfloor, \ldots, \lfloor \frac{m}{n} \rfloor]$ is consequently a kernel vector of the $\langle n, m, \ell, u\rangle$-GSB task.

To solve the asymmetric $\langle n, m, \vec{\ell}, \vec{u}\rangle$-GSB task, we first consider the set of output vectors $\mathcal{O}$. We then order these vectors in the same, deterministic way, and pick the first one. Let V be this vector of the $\langle n, m, \vec{\ell}, \vec{u}\rangle$-GSB task. We use then the same vector V for all processes. Let dec_i be the value obtained by process p_i in the $\langle n, n, 1, 1\rangle$-GSB task. A process p_i then considers $V[dec_i]$ entry as its output $output_i$ with respect to the $\langle n, m, \vec{\ell}, \vec{u}\rangle$-GSB simulated task. Because the $\langle n, n, 1, 1\rangle$-GSB task has a single kernel vector $[1, \ldots, 1]$, it follows that each entry of V is chosen by only a single process.This satisfies the specification of the $\langle n, m, \vec{\ell}, \vec{u}\rangle$-GSB task, which concludes the proof of the theorem. $\square_{Theorem\ 8}$

5.2 Easiest GSB tasks: Solvability of GSB tasks with no communication

This section identifies the easiest of all the GSB tasks, namely those that are solvable with no communication at all. It is easy to see that any feasible GSB task where $m = 1$ is solvable without any communication (a single value can be decided). The next theorem characterizes the communication-free GSB tasks when $m > 1$.

Theorem 9 *Consider an $\langle n, m, \ell, u\rangle$-GSB task T where $m > 1$. Then, T is solvable with no communication if and only if $(\ell = 0) \wedge (\lceil \frac{2n-1}{m} \rceil \leq u)$.*

Proof Let us first assume $\ell = 0$ and $u = \lceil \frac{2n-1}{m} \rceil$ (increasing u makes the problem even easier). Recall that the identities of the processes are taken from $1..2n-1$. Let us deterministically partition the $2n-1$ identities into m groups, $G_1, \ldots, G_m$, so that no group has more than $\lceil \frac{2n-1}{m} \rceil$ elements and no group has less than $\lfloor \frac{2n-1}{m} \rfloor$ elements. Let δ be the deterministic function that maps identities in group G_i to i(the partitioning and δ are known by every process). To solve T with no communication, each process p_i outputs $\delta(id_i)$ and we have that each value $x \in [1..m]$ is decided by at most $\lceil \frac{2n-1}{m} \rceil$ processes.

For the other direction, let us first consider an $\langle n, m, \ell, u \rangle$-GSB task T with $m > 1$ and $u < \lceil \frac{2n-1}{m} \rceil$. Suppose, by way of contradiction, that there is an algorithm $\mathcal{A}$ that solves T with no communication. The algorithm implies a decision function δ that assigns to each identity x in $1..2n-1$, an output value $\delta(x)$ in $1..m$. The value $\delta(x)$ is the decision produced by a process when it starts with identity x, without any communication. Define groups G_i by putting in the same group identities x, x' whenever $\delta(x) = \delta(x')$. For any partition of the set of identities, the size of the biggest group is at least $\lceil \frac{2n-1}{m} \rceil$. The task specification requires that for each i, $|G_i| \leq u < \lceil \frac{2n-1}{m} \rceil$, which is impossible.

Let us now consider an $\langle n, m, \ell, u \rangle$-GSB task T with $m > 1$ and $\ell > 0$. For any partition of the set of identities, as $m \geq 2$, the size of the smallest group is at most $\lfloor \frac{2n-1}{m} \rfloor \leq n-1$. The task specification requires that, for each i, $|\{p_j \mid \delta(id_j) = i\}| \geq \ell \geq 1$. Because there are $n-1$ identities not corresponding to any process and the size of the smallest group obtained from the partitioning is at most $n-1$, it follows that it is possible that no process belongs to some group, which concludes the proof. $\square_{Theorem\ 9}$

Let us call *x-bounded homonymous renaming* the $\langle n, \lceil \frac{2n-1}{x} \rceil, 0, x \rangle$-GSB task. This task can easily be solved: process p_i decides the value $\lceil \frac{id_i}{x} \rceil$.

Corollary 2 *The x-bounded renaming $\langle n, \lceil \frac{2n-1}{x} \rceil, 0, x \rangle$-GSB task is solvable with no communication.*

The next corollary is an immediate consequence of Theorem 9 when $m = 2$ and $\ell = 1$.

Corollary 3 *The WSB $\langle n, 2, 1, n-1 \rangle$-GSB task is not solvable without communication.*

When $m = 2n-1$ in Theorem 9, we have the trivial $\langle n, 2n-1, 0, 1 \rangle$-GSB, which is actually the classical (non-adaptive) $(2n-1)$-renaming problem for which many solutions have been proposed (e.g., [5, 8, 15]; see [18] for an introductory survey). In our setting (where according to Theorem 1, we have $\forall i : id_i \in [1..2n-1]$), to solve $\langle n, 2n-1, 0, 1 \rangle$-GSB task each process outputs its own identity.

Interestingly, as mentioned later, when considering $m = 2n-2$ and the $\langle n, 2n-2, 0, 1 \rangle$-GSB task, things become much more interesting. This task may or may not be wait-free solvable, depending on the value of n. The proof of the following corollary is obtained by replacing $(2n-1)$ by $2(n-k)$ in the proof of Theorem 9.

Corollary 4 *The k-WSB $\langle n, 2, k, n-k \rangle$-GSB task is solvable without communication from $2(n-k)$-renaming.*

5.3 Hierarchy results, GSB tasks of intermediate difficulty

While the renaming $\langle n, 2n-1, 0, 1 \rangle$-GSB task is solvable with no communication, the renaming $\langle n, 2n-2, 0, 1 \rangle$-GSB task is not wait-free solvable, except for some special values of n [16, 17]. Interestingly, [29] shows that $\langle n, 2n-2, 0, 1 \rangle$-GSB and the WSB $\langle n, 2, 1, n-1 \rangle$-GSB task are wait-free equivalent: any of $\langle n, 2, 1, n-1 \rangle$-GSB and $\langle n, 2n-2, 0, 1 \rangle$-GSB can be solved in the system model $\mathcal{ASM}_{n,n-1}[\emptyset]$ enriched with a solution to the other task.

Let us remember that a set of integers $\{n_i\}$ is prime if $\gcd\{n_i\} = 1$.

Theorem 10 *Let $m > 1$. If the set $\{\binom{n}{i} : 1 \leq i \leq \lfloor \frac{n}{2} \rfloor\}$ is not prime, then $\langle n, m, 1, u \rangle$-GSB is not wait-free solvable, $\forall u$.*

Proof For any $m > 1$, the $\langle n, m, 1, (n-m+1) \rangle$-GSB task solves the WSB $\langle n, 2, 1, n-1 \rangle$-GSB task: the processes decide the output of the $\langle n, m, 1, n \rangle$-GSB task modulo 2. It has been shown in [29] that WSB and $(2n-2)$-renaming are equivalent. It has been shown in [17] that $(2n-2)$-renaming is not read/write wait-free solvable when $\{\binom{n}{i} : 1 \leq i \leq \lfloor \frac{n}{2} \rfloor\}$ is not prime. The $\langle n, m, 1, (n-m+1) \rangle$-GSB task is then not wait-free solvable either. Moreover, if $m > n$, the $\langle n, m, 1, (n-m+1) \rangle$-GSB task is not feasible. Let us then consider the case in which $n \geq m > 1$. It follows from Theorem 3 that, $\forall m \leq n$, the $\langle n, m, 1, (n-m+1) \rangle$-GSB task is a feasible ℓ-anchored task. Thus, $\forall\, u \geq (n-m+1)$, the $\langle n, m, 1, u \rangle$ and $\langle n, m, 1, (n-m+1) \rangle$-GSB tasks are synonyms. On another side, it follows from Lemma 4 that, $\forall\, n, m, \ell$ and $u' \geq u$, the $\langle n, m, \ell, u \rangle$-GSB task T and the $\langle n, m, \ell, u' \rangle$-GSB task T' are such that $S(T) \subseteq S(T')$. Thus if the $\langle n, m, 1, (n-m+1) \rangle$-GSB task is not wait-free solvable, then the $\langle n, m, 1, u \rangle$-GSB task is not wait-free solvable either for any $u \geq (n-m+1)$, which concludes the proof of the theorem. $\square_{Theorem\ 10}$

Now, consider the election asymmetric GSB task: one process decides 1, while $n-1$ processes decide 2. The outputs vectors of this task are contained in the output vectors of the WSB $\langle n, 2, 1, n-1 \rangle$-GSB task, and hence, election trivially solves WSB. Moreover, election is strictly stronger than WSB because election is not wait-free solvable (see below), while WSB is solvable for (infinitely many) values of n [17].

Theorem 11 *The election GSB task is not wait-free solvable.*

Proof Assume for contradiction there is a wait-free algorithm solving election. By Lemma 2 we can assume the algorithm is comparison based. This implies that a process running solo, always decides the same binary value, independently of its input name (and of its index, as the algorithm is index-independent).

Consider as in [10, 17, 35] the complex of the algorithm. This complex is made of $(n-1)$-simplexes (sets of size n), and all their faces (subsets). Each 0-simplex is a vertex, labeled with the local state of one of the processes. Each $(n-1)$-simplex corresponds to a set of executions of the algorithm that are indistinguishable to the processes, and where every process has decided a binary value, solving election. Thus, the n vertices of every $(n-1)$-simplex are labeled with distinct processes, and each vertex is also labeled with the local state of the process at the end of an execution corresponding to that simplex. These local states include the value decided by the process, and are such that, in every $(n-1)$-simplex, exactly one vertex is labeled 1 and $n-1$ vertices are labeled 2, as election is solved.

Moreover, as in the previous papers (where the following properties are proved), we may consider only the subset of executions of the algorithm corresponding to immediate snapshots. This implies the complex is a pseudo-manifold. That is, every $(n-2)$-simplex is contained in either one or two $(n-1)$-simplexes. Also, the complex is connected: there is a path connecting any two $(n-1)$-simplexes, consisting of a sequence of $(n-1)$-simplexes, where each consecutive two $(n-1)$-simplexes intersect in an $(n-2)$-simplex.

We now prove that each process must decide the same binary value in every (immediate snapshot) execution of algorithm. Consider any internal $(n-2)$-simplex, contained in two $(n-1)$-simplexes. Let v_1, v_2 be the two vertices in these simplexes, that do not belong to the $(n-2)$-simplex. These correspond to the same process, say p_i. Notice that p_i decides the same binary value, b, in both v_1 and v_2, because every $(n-1)$-simplex is labeled with exactly one 1. By connectivity of the algorithm complex, every vertex of the complex corresponding to p_i, the decision of p_i is b.

In particular, the vertex v_i, corresponding to the solo execution by p_i, is also labeled with the decision b. Therefore, we see that the n vertices $v_1, \ldots, v_n$, corresponding to the solo executions of the n processes, are labeled with decision values, such that exactly one decides 1 and $n-1$ decide 2. This contradicts the assumption that the algorithm is comparison-based. $\square_{Theorem\ 11}$

The next corollary follows from the fact that leader election is not wait-free solvable and perfect renaming is universal for the family of GSB tasks.

Corollary 5 *The perfect renaming GSB task is not wait-free solvable.*

6 From a slot task to a renaming task

This section presents a simple algorithm that solves the $(n+1)$-renaming task ($\langle n, n+1, 0, 1\rangle$-GSB task) in the system model $\mathcal{ASM}_{n,n-1}[\langle n, n-1, 1, n\rangle$-GSB]. The underlying object solving the $\langle n, n-1, 1, n\rangle$-GSB task is denoted KS. It provides the processes with a single operation denoted $\mathsf{slot\_request}_{n-1}()$ whose semantics has been described in Section 3.2 (namely, each value x, $1 \leq x \leq n-1$, is decided by at least one process).

Shared objects In addition to KS, the processes cooperate through a snapshot object denoted $STATE[1..n]$. Each register $STATE[i]$ is initialized to $\perp$ and can be written only by p_i. Process p_i writes into it a pair of integers $\langle my\_slot_i, id_i\rangle$ (where my_slot_i is the slot number it obtains from KS and id_i (its identity). A process obtains the value of the snapshot object by invoking $STATE.\mathsf{snapshot}()$.

```
operation new_name():
(01) my_slot_i ← KS.slot_request_{n-1}();
(02) STATE[i] ← ⟨my_slot_i, id_i⟩; (slot_i[1..n], ids_i[1..n]) ← STATE.snapshot();
(03) if (∀j ≠ i : slot_i[j] ≠ my_slot_i)
(04)    then return(my_slot_i)
(05)    else let j ≠ i such that slot_i[j] = my_slot_i;
(06)         if (id_i < ids_i[j]) then return(n) else return(n + 1) end if
(07) end if.
```

Figure 2: Solving $(n+1)$-renaming in $\mathcal{ASM}_{n,n-1}[\langle n, n-1, 1, n\rangle$-GSB] (code for p_i)

Process behavior Each process p_i manages two local arrays denoted $slot_i[1..n]$ and $ids_i[1..n]$. These arrays are used to keep the values read from the two fields of the snapshot object $STATE$. The algorithm for process p_i is depicted in Figure 2. It is made up of two parts.

- A process p_i first acquires a slot number (line 01). Then it writes its attributes (slot number and identity) in $STATE[i]$ and reads the snapshot object to obtain an "atomic" global view of all the attributes that have been posted (line 02 where the read is denoted $\mathsf{snapshot}()$).

- Then process p_i determines its new name which is its slot number if it sees no other process with the same slot number (lines 03-04). In the other case, it follows from the properties of the KS object that there is a single process p_j that has obtained the same slot number s (line 05). Processes p_i and p_j are consequently competing for a new name. Moreover, it is possible that p_j has already considered slot s as its new name. Process p_i solves this conflict according to the order on its identity and p_j's identity: if p_i's identity is smaller, it considers n as its new name, otherwise it considers $n+1$ as its new name (line 06).

Theorem 12 *The algorithm described in Figure 2 solves the $(n+1)$-renaming task from any solution to the $(n-1)$-slot task.*

Proof The wait-freedom property and the fact that the new names belong to the set $\{1,\ldots,n+1\}$ follow directly from the text of the algorithm. Hence, we only focus on the proof that no two process obtain the same new name.

Due to the property of the KS object that assigns $n-1$ slots to n processes, it follows that $n-2$ processes are assigned distinct slots and those are in $[1..n-1]$. Let p_x and p_y be the processes that are assigned the same slot s. The proof follows from the fact that the snapshot invocations are totally ordered. There are tow cases.

- The snapshot value obtained by p_x is such that $STATE[y]=\perp$. In that case, p_x returns s as its new name. Moreover, the snapshot value obtained by p_y will be such that $STATE[x]=s$. Hence, p_y will obtain the new name n or $n+1$ according to the values of id_x and id_y.
- The snapshot values obtained by p_x are p_y are such that both $STATE[x]$ and $STATE[y]$ are equal to s. In that case, both execute lines 05-06, from which it follows that they obtain new names n and $n+1$ according to the order on id_x and id_y.

□ *Theorem 12*

Towards a general algorithm It is well-known that the $(2n-2)$-renaming task and the weak symmetry breaking task are equivalent (e.g., [18]). As, the weak symmetry breaking task and the 2-slot task are the same task, it follows that the $(2n-2)$-renaming task and the 2-slot task are equivalent.

More generally, when considering the more general problem of finding an algorithm that solves the $(2n-k)$-renaming task from any solution to the k-slot task, the algorithm in Figure 2 is a specific answer for $k=n-1$, while the equivalence between weak symmetry breaking and the 2-slot task is a specific answer for $k=2$.

As indicated in the Introduction, answering the question "Is there a general algorithm that solves $(2n-k)$-renaming from the k-slot task and more generally are the $(2n-k)$-renaming task and the k-slot task equivalent?" constitutes a difficult but promising challenge.

7 To conclude: a few GSB-related open problems

In addition to the previous question, many interesting questions concerning the family of GSB tasks remain open. Here are a few. Is perfect renaming the only universal GSB task? What is the structure of the hierarchy of GSB tasks? Namely, is it a partial order, a total order? Are there incomparable tasks? Which ones? Etc.

References

[1] Afek Y., Attiya H., Dolev D., Gafni E., Merritt M. and Shavit N., Atomic Snapshots of Shared Memory. *Journal of the ACM*, 40(4):873-890, 1993.

[2] Afek Y., Gafni E. and Lieber., Tight Group Renaming on Groups of Size g Is Equivalent to g-Consensus. *Proc. 23rd Int'l Symposium on Distributed Computing (DISC'09)*, Springer Verlag LNCS #5805, pp. 111-126, 2009.

[3] Afek Y., Gafni E., Rajsbaum S., Raynal M. and Travers C., The k-Simultaneous Consensus Problem. *Distributed Computing*, 22:185-195, 2010.

[4] Afek Y., Gamzu I., Levy I., Merritt M., and Taubenfeld G., Group Renaming. *Proc. 12th International Conference on Principles of Distributed Systems (OPODIS'08)*, Springer Verlag LNCS #5401, pp. 58-72, 2008.

[5] Afek Y. and Merritt M., Fast, Wait-Free $(2k-1)$-Renaming. *Proc. 18th ACM Symposium on Principles of Distributed Computing (PODC'99)*, ACM Press, pp. 105-112, 1999.

[6] Angluin D., Local and Global Properties in Networks of Processors. *Proc. 12th ACM Symposium on Theory of Computing (STOC'80)*, ACM Press, pp. 82-93, 1980.

[7] Attiya H., Bar-Noy A., Dolev D., Peleg D. and Reischuk R., Renaming in an Asynchronous Environment. *Journal of the ACM*, 37(3):524-548, 1990.

[8] Attiya H. and Fouren A., Polynomial and Adaptive Long-lived $(2p-1)$-Renaming. *Proc. 14th Int'l Symposium on Distributed Computing (DISC'00)*, Springer Verlag LNCS #1914 , pp.149-163, 2000.

[9] Attiya H., Gorbach A. and Moran S., Computing in Totally Anonymous Asynchronous Shared Memory Systems. *Information and Computation*, 173(2):162–183, 2002.

[10] Attiya H. and Rajsbaum S., The Combinatorial Structure of Wait-Free Solvable Tasks, *SIAM Journal of Computing,* 31(4):1286-1313, 2002.

[11] Attiya H. and Welch J., Distributed Computing: Fundamentals, Simulations and Advanced Topics, (2d Edition), *Wiley-Interscience*, 414 pages, 2004.

[12] Borowsky E., Gafni E., Lynch N. and Rajsbaum S., The BG Distributed Simulation Algorithm. *Distributed Computing*, 14(3): 127-146, 2001.

[13] Burns, J., Symmetry in Systems of Asynchronous Processes. *22nd IEEE Symposium on Foundations of Computer Science (FOCS'81)*, IEEE Computer Press, 169-174, 1981.

[14] Borowsky E. and Gafni E., Generalized FLP Impossibility Result for t-Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on Theory of Computing (STOC'93)*, ACM Press, pp. 91-100, 1993.

[15] Borowsky E. and Gafni E., Immediate Atomic Snapshots and Fast Renaming. *Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC'93)*, ACM Press, pp. 41-51, 1993.

[16] Castañeda A., A Study of the Wait-free Solvability of Weak Symmetry Breaking and Renaming. PhD Thesis, Posgrado en Ciencia e Ingeniería de la Computación, UNAM, Mexico, December 2010.

[17] Castañeda A. and Rajsbaum S., New Combinatorial Topology Upper and Lower Bounds for Renaming. *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*, ACM Press, pp. 295-304, 2008.

[18] Castañeda A., Rajsbaum S. and Raynal M., The Renaming Problem in Shared Memory Systems: an Introduction. *Tech Report 1960*, IRISA, Université de Rennes (F), 29 pages, 2010. Submitted to publication.

[19] Chalopin J. and Métivier Y., On the Power of Synchronization Between two Adjacent Processes. Distributed Computing 23(3): 177-196, 2010.

[20] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, 1996.

[21] Chaudhuri S., More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation*, 105(1):132-158, 1993.

[22] Dijkstra, E.W., Solution of a Problem in Concurrent Programming Control. *Communications of the ACM,* 8(9):569, 1965.

[23] Dolev D., Lynch N., Pinter S., Stark E. and Weihl W. Reaching Approximate Agreement in the Presence of Faults. *Journal of the ACM*, 33(3):499–516, 1986.

[24] Dinitz Y., Moran S. and Rajsbaum S., Bit complexity of Breaking and Achieving Symmetry in Chains and Rings. *Journal of the ACM*, 55(1), article 3, 32 pages, 2008.

[25] Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.

[26] Gafni E., The 01-Exclusion Families of Tasks. *Proc. 12th In'l Conference on Principles of Distributed Systems (OPODIS'08*, Springer Verlag LNCS #5401, pp. 246-258, 2008.

[27] Gafni G., Mostéfaoui A., Raynal M. and Travers C., From Adaptive Renaming to Set Agreement. *Theoretical Computer Science*, 410(14-15): 1328-1335, 2009.

[28] Gafni E. and Rajsbaum S., Musical Benches. *19th International Symposium on Distributed Computing (DISC'05)*, Springer Verlag LNCS #3724, pp. 63-77, 2005.

[29] Gafni E., Rajsbaum S. and Herlihy M., Subconsensus Tasks: Renaming is Weaker Than Set Agreement. *Proc. 20th Int'l Symposium on Distributed Computing (DISC'06)*, Springer Verlag LNCS #4167, pp.329-338, 2006.

[30] Gafni E., Rajsbaum S., Raynal M. and Travers C., The Committee Decision Problem. *Proc. Latin American Theoretical Informatics Symposium (LATIN'06)*. Springer Verlag LNCS #3887, pp. 502-514, 2006.

[31] Herlihy M.P., Wait-Free Synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149, 1991.

[32] Herlihy M. and Rajsbaum S. The Decidability of Distributed Decision Tasks. *Proc. 29th ACM Symposium on Theory of Computing (STOC'97)*, ACM Press, pp. 589-598, 1997.

[33] Herlihy M. and Rajsbaum S. A Classification of Wait-Free Loop Agreement Tasks. *Theoretical Computer Science*, 291(1):55-77, 2003.

[34] Herlihy M. and Rajsbaum S. The topology of Shared-Memory Adversaries. *Proc. 29th ACM Symposium on Principles of Distributed Computing (PODC'10)*, ACM Press, pp. 105-113, 2010.

[35] Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923,, 1999.

[36] Jayanti, P., and Toueg, S. Wakeup under Read/Write Atomicity. *Proc. 4th Int'l Workshop on Distributed Algorithms (WDAG'90)*, Springer Verlag LNCS #486, pp. 277–288, 1990.

[37] Lamport. L., The Part-time Parliament. *ACM Transactions on Computer Systems*, 16(2):133-169, 1998.

[38] Mazurkiewicz A., Distributed Enumeration. *Information Processing Letters* 61:233239, 1997.

[39] Moran, S., and Wolfsthal, Y., An extended Impossibility Result for Asynchronous Complete Networks. *Information Processing Letters* 26:141-151, 1987.

[40] Raynal M., Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems. *Morgan & Claypool Publishers*, 251 pages, 2010 (ISBN 978-1-60845-293-4).

[41] Styer, E., and Peterson, G. L., Tight Bounds for Shared Memory Symmetric Mutual Exclusion Problems. *Proc. 8th ACM Symposium on Principles of Distributed Computing (PODC'89)*, ACM Press, pp. 177-192, 1989.

[42] Saks M. and Zaharoglou F., Wait-Free k-Set Agreement Is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.