Enriching the reduction map of sub-consensus tasks

Armando Castañeda, Damien Imbs, Sergio Rajsbaum, Michel Raynal

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 1976 – Avril 2011

Enriching the reduction map of sub-consensus tasks

Armando Castañeda[*] , Damien Imbs[**] , Sergio Rajsbaum[***] , Michel Raynal[****]
armando.castanedar@inria.com, damien.imbs@irisa.fr, rajsbaum@math.unam.mx, raynal@irisa.fr

Abstract: Understanding the relative computability power of tasks, in the presence of asynchrony and failures, is a central concern of distributed computing theory. In the *wait-free* case, where the system consists of n processes and any of them can fail by crashing, substantial attention has been devoted to understanding the relative power of the *subconsensus* family of tasks, which are too weak to solve consensus for two processes. The first major results showed that set agreement and renaming (except for some particular values of n) cannot be solved wait-free in read/write memory. Then it was proved that renaming is strictly weaker than set agreement (when n is odd).

This paper considers a natural family of subconsensus tasks that includes set agreement, renaming and other generalized symmetry breaking (GSB) tasks. It extends previous results, and proves various new results about when there is a reduction and when not, among these tasks. Among other results, the paper shows that there are incomparable subconsensus tasks.

Key-words: Decision task, Distributed computability, Problem hierarchy, Renaming problem, k-Set agreement, Symmetry Breaking, Wait-freedom.

Enrichir la carte de réduction des tâches sous-consensus

Résumé : *Ce rapport enrichit la carte de réduction des tâches sous-consensus*

Mots clés : *Tâche de décision, Calculabilité distribuée, Hiérarchie de problèmes, Renommage, k-Accord ensembliste, Cassage de symétrie, Sans-attente.*

---

[*] Projet ASAP: équipe commune avec l'INRIA, le CNRS, l'université Rennes 1 et l'INSA de Rennes
[**] 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

A *task* is the distributed computing equivalent of the notion of a *function* encountered is sequential computing. In a task, each input is a vector, whose entries are distributed to the processes, and after communicating with each other, processes decide on local outputs, that together form a legal output vector respecting the task specification. In an asynchronous system, a protocol solves a task *wait-free* if any process that continues to run will halt with an output value in a fixed number of steps, regardless of delays or crashes of other processes.

Understanding the relative computability power of tasks, in presence of asynchrony and failures, is a central concern of distributed computing theory. Given two tasks, can one be used to implement the other, or are they incomparable? To this end, an important line of research consists in defining relevant families of tasks, designing reductions from tasks to other tasks, proving when such reductions are impossible, and looking for tasks that are universal for the family.

The consensus hierarchy Measuring the relative power of tasks using *consensus numbers* [19] has been very fruitful. A task has consensus number x if it is powerful enough to wait-free implement *consensus* [11] in a system of x processes but too weak to implement it in a system of $x+1$ processes. If it can implement consensus for any number of processes, its consensus number is $+\infty$. If a task can solve consensus for n processes, it is *universal* in a n-process system [19], in the sense that it can be used to solve any other task in such a system. The *consensus hierarchy* implied by this result describes the relative power of a large family of tasks. As shown in [19], read/write registers have consensus number 1, test&set, queues, stacks have consensus number 2, etc., until tasks such as compare&swap or LL/SC whose consensus number is $+\infty$. The consensus hierarchy provides us with a simple way to know if a given task is computationally stronger than another in the presence of asynchrony and any number of process crashes.

Sub-consensus tasks Substantial attention has been devoted to understanding the relative power of the *subconsensus* family of tasks, which are too weak to solve consensus for two processes, and yet, very little is known. Subconsensus tasks have a fine structure, inaccessible by consensus-based analysis. In summary, the only hierarchy results known are the following set agreement and renaming results. First, while both tasks have consensus number 1 (as read/write registers), they cannot be implemented from read/write registers, except for some particular values of n, for which renaming can be implemented from read/write registers. Second, it was proved that renaming is strictly weaker than set agreement, when n is odd. We now describe in more detail these results.

In the (n,k)-*set agreement* task [9], n processes have to agree on at most k different values. It is a weakening of consensus, and when $k=1$, it is equal to consensus. The first major result about subconsensus tasks was that (n,k)-set agreement cannot be implemented from read/write registers, even when $k=n-1$ [3, 22, 26], and lead to the discovery of a deep connection between distributed computing and topology. Later on, the structure of the set agreement family of tasks was identified to be a partial order, and it was shown that (n,k)-set agreement cannot be used to solve consensus among two processes, even when $k=2$, e.g. [10, 20].

When solving M-*renaming* [1] the processes have to decide distinct names from a name space whose size M is as small as possible. Initially it was proved that $(n+1)$-renaming, cannot be wait-free solved in a read/write system [1]. It took a substantial use of topology to show that (except for some specific values of n [7]), M-renaming can be implemented out of read/write registers if and only if $M \geq 2n-1$ [22].

Set agreement and renaming appear to be quite dissimilar tasks, one being about agreement while the other is about symmetry breaking. Thus it was surprising to know [16, 17] that $(n,n-1)$-set agreement can be used to implement $(2n-2)$-renaming, while the opposite is impossible, when n is odd. The result was first proved in [17] in a natural round-by-round iterated computational model (IM) (e.g. see [25]), and then extended to the usual read/write model in [16] using a simulation.

The family of generalized symmetry breaking tasks Until recently, the main hierarchy results for subconsensus tasks where about (n,k)-set agreement and M-renaming, and only for the particular case of $k=n-1$, $M=2n-2$, and n odd. In an effort to expand these results, a conceptual framework has recently been introduced in [23], to investigate the family of *generalized symmetry breaking* (GSB) tasks. The notation $\langle n,m,\ell,u\rangle$-GSB is used to denote the tasks on n processes for m possible decision values, $[1..m]$, where each value has to be decided by at least ℓ and at most u processes[1] Some examples of GSB tasks are the following. M-renaming is nothing else than the $\langle n,M,0,1\rangle$-GSB task. Weak symmetry breaking [17] is the $\langle n,2,1,n-1\rangle$-GSB task. A new task is k-*slot* [23], the $\langle n,k,1,n\rangle$-GSB task.

Among other results, it is shown in [23] that *perfect renaming*, i.e., $\langle n,n,1,1\rangle$-GSB, is universal for the whole family of GSB tasks, in the sense that any of these tasks can be solved from an algorithm solving the $\langle n,n,1,1\rangle$-GSB task.

Contents of the paper The aim of this paper is to expand our knowledge about the fine structure of subconsensus tasks, enriching the map of reductions among set agreement, renaming and other GSB tasks. The map of Figure 1 summarizes our new results, as well as previous results. An arrow with a black dot represents a new reduction; if it has a cross it is an impossibility result. As any GSB task on n processes is solvable from $\langle n,n,1,1\rangle$-GSB, the corresponding arrows are not depicted.

[1] Such a task is called *symmetric* in [23]. In the *non-symmetric* GSB tasks, ℓ and u are replaced by two m-size vectors $\vec{\ell}$ and $\vec{u}$ indicating that $\forall x \in [1..m]$, x must be decided at least $\vec{\ell}[x]$ and at most $\vec{u}[x]$ times.

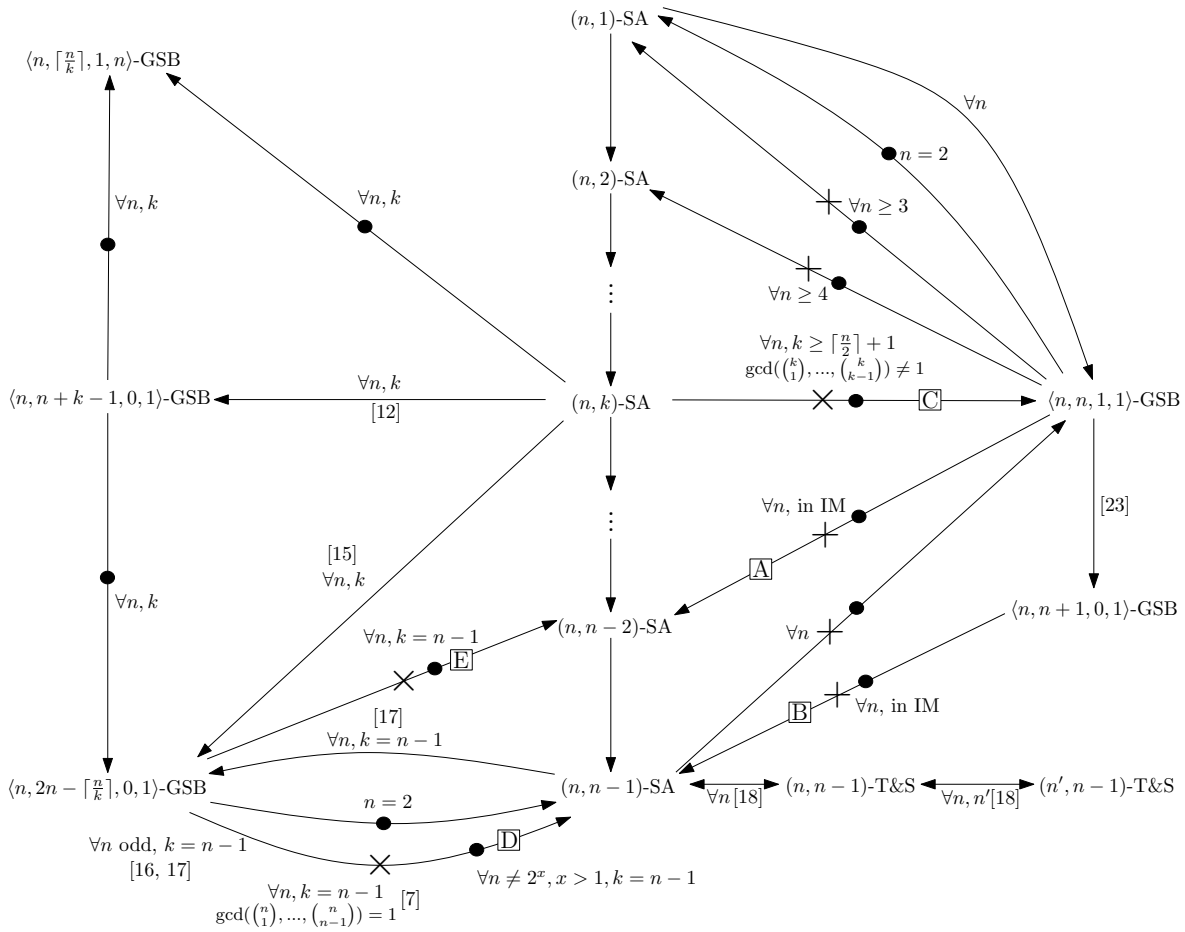

Figure 1: A map relating (n, k)-set agreement tasks and GSB tasks.

Perhaps our main result is showing that two apparently very different kinds of subconsensus tasks, perfect renaming and $(n, n-2)$-set agreement, an agreement and a symmetric breaking, are incomparable. We single out from the map of Figure 1 the following results, arrows labeled A,B,C, D and E.

- Arrow A : Perfect renaming cannot implement $(n, n-2)$-set agreement in the IM (Theorem 7).
- Arrow B: $(n+1)$-renaming (i.e., the most powerful non-perfect renaming problem) cannot implement $(n, n-1)$-set agreement in the IM (Theorem 6).
- Arrow C: A characterization of values of k for which (n, k)-set agreement cannot implement perfect renaming (Theorem 9).
- Arrow D: A partial answer to the question left open in [17] where it was shown that $(2n-2)$-renaming cannot implement $(n, n-1)$-set agreement when n is odd. This result is extended to all $n \neq 2^x, x > 1$ (Theorem 4), i.e., it is extended for infinitely many values of n. However, let us observe that for $n \geq 3$, arrow B closes the open question in [17], in the IM, since obviously $(n+1)$-renaming can implement $(2n-2)$-renaming, for $n \geq 3$. In fact, note that arrow B strengthens significantly this result showing that, in the IM, even $(n+1)$-renaming cannot implement $(n, n-1)$-set agreement.
- Arrow E: $(2n-2)$-renaming cannot implement $(n, n-2)$-set agreement, for any n (Theorem 5). Of course all cases $n \neq 2^x$, $x > 1$, are implied by arrow D, however, the proof of this result is simple and does not use any topological argument.

When considering the map of Figure 1 let us notice that the right part of the bottom line is due to [18] where it is shown that (a) $(n, n-1)$-set agreement and $(n, n-1)$-test&set are equivalent and (b) $(n, n-1)$-test&set and $(n', n-1)$-test&set are also equivalent for any $n' \geq n$.

Related work A hierarchy of subconsensus tasks has been defined in [14] where a problem P belongs to class k if k is the smallest integer such that P can be wait-free solved in an n-process asynchronous read/write system enriched with (n, k)-set agreement objects.

Also, [21] studies the hierarchy of *loop agreement* subconsensus tasks, under a restricted implementation notion, and identify an infinite hierarchy, where some loop agreement tasks are incomparable.

The M-renaming problem considered in this paper is different from the *adaptive renaming* version, where the size of the output name space depends on the actual number of processes that participate in a given execution, and not on the total number of processes of the system, n. While the consensus number of perfect adaptive renaming is known to be 2 [8], in this paper we consider the relative power of non-adaptive renaming. Let us recall that in a system with n processes, adaptive $(2p - \lceil \frac{p}{n-1} \rceil)$-renaming is equivalent to $(n, n-1)$-set agreement [18] where p, $1 \leq p \leq n$, denotes the number of participating processes. It is shown in [15] that (n, k)-set agreement can be solved from adaptive $(p + k - 1)$-renaming. Combined with this paper, this emphasizes an important difference between adaptive and non-adaptive renaming. Let us also notice that test&set and its variants are adaptive while GSB tasks are not.

Roadmap The paper is composed of 5 sections. Section 2 presents formally the model, the notion of a task and the notations used in the paper. Then, Section 3 focuses on the new arrows from GSB tasks to (n, k)-set Agreement, while Section 4 focuses on the new arrows in the other direction. Finally, Section 5 concludes the paper.

2 Model, tasks and notation

2.1 Base read/write wait-free computation model

Due to space limitations and the fact that this model is widely used in the literature, we do not explain it in detail here. A detailed description of this model is given in [23]. Nevertheless, we restate carefully some aspects of this model because we are interested in a *comparison-based* and an *index-independent* solvability notion that are not as common.

Read/write wait-free system model This paper considers the usual asynchronous, wait-free shared memory system where at most $n-1$ out of n processes p_1, ..., p_n can fail by crashing. 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 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.

The memory is made up of single-writer/multi-reader registers. The subscript i (used in p_i) is called the *index* of p_i. Indexes are used only for addressing purposes (this is formalized below).

The algorithms designed for this computation model have to work despite up to $n-1$ process crashes. In some sections, in addition to registers, processes are allowed to cooperate through certain objects that implement some task T.

Identities Each process p_i has an identity denoted id_i that is kept in $input_i$. 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: $i \neq j \Rightarrow input_i \neq input_j$.

A process 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, N and the fact 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 $\pi()$ of the process indexes. 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 $\mathcal{A}$.

Let a permutation $\pi()$ such that $\pi(i) = j$. The index-independence ensures that p_j behaves in r_π exactly as p_i behaves in r: it decides the same thing 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.

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 $i_1 < i_2 < \cdots < i_n$ of a run r of $\mathcal{A}$ and any other ordered inputs $j_1 < j_2 < \cdots < j_n$. 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.

2.2 Tasks

Task A one-shot decision problem is specified by a *task* $(\mathcal{I}, \mathcal{O}, \Delta)$, that consists of a finite 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 [22]). All vectors are n-dimensional.

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 a local variable $input_i$) and each non-faulty process eventually decides on an output value by writing it to a local 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) for each process p_i that decides we have $J[i] = output_i$.

The (n, k)-set agreement (SA) task Each process p_i is assumed to propose a value and each correct process has to decide a value such that the following properties are satisfied.

- Termination. Each process decides a value.
- Validity. A decided value is a proposed value.
- Agreement. At most k different values are decided.

The (n, k)-test and set (T&S) task Each process p_i is required to decide a value such that the following properties are satisfied. (The instance $k = 1$ does correspond to the usual Test&set object.)

- Termination. Each process decides a value.
- Validity. Each process decides 0 (winer) or 1 (looser).
- Agreement. At least one and at most k processes decide the value 0.

The $\langle n, m, \ell, u\rangle$-GSB tasks Each process starts with a distinct identity from a set $[1..N]$. Each correct process has to decide a value such that the following properties are satisfied.

- Termination. Each correct process decides a value.
- Validity. A decided value belongs to $[1..m]$.
- Symmetric agreement. Each value $v \in [1..m]$ is decided by at least ℓ and at most u processes, in executions where all decide.

We consider only feasible tasks, i.e., tasks for with $\Delta(I) \neq \emptyset$. It is easy to see that $m \times \ell \leq n \leq m \times u$ is a necessary and sufficient condition for the $\langle n, m, \ell, u\rangle$-GSB task to be feasible. The structure, complexity and computability issues of the universe of $\langle n, m, \ell, u\rangle$-GSB tasks is investigated in [23].

As already indicated, the $\langle n, m, 0, 1\rangle$-GSB task is the (non-adaptive) m-renaming problem while the new $\langle n, k, 1, n - k + 1\rangle$-GSB task (which we call k-slot task) is a generalization of the WSB (weak symmetry breaking) task (which is the $\langle n, 2, 1, n - 1\rangle$-GSB task). Other new GSB tasks are described in [23].

Notation 1 *Let T and T' be two tasks. $T \rightarrow T'$ means that there is an algorithm that solves task T' in the read/write wait-free model enriched with objects that solve task T. $T \leftrightarrow T'$ is a shortcut for $T \rightarrow T' \wedge T' \rightarrow T$ (T and T' are equivalent). $T \not\rightarrow T'$ means that there is no algorithm that solves task T' in the read/write wait-free model enriched (only) with objects that solve task T.*

2.3 An iterated model

Attempts at unifying different read/write distributed computing models have restricted their attention to a subset of *round-based* executions. The approach introduced in [5] generalizes these attempts by proposing an *iterated* model in which processes execute an infinite sequence of rounds, and in each round communicate through a specific object that provides processes with a single operation denoted write_snapshot().

One-shot write-snapshot object When invoked by a process p_i, the semantics of the write_snapshot() operation consists of executing a write, and immediately a snapshot operation. More precisely, it is defined by the following properties, where v_i is the value written by p_i and sm_i, the value (or *view*) it gets back from the operation. A view sm_i is a set of pairs (id_k, v_k), where v_k is the value written by p_k in the object. (Let $sm_i = \emptyset$ if the process p_i never invokes write_snapshot().) These properties are:

- Self-inclusion. $\forall i : (id_i, v_i) \in sm_i$.
- Containment. $\forall i, j : sm_i \subseteq sm_j \ \vee \ sm_j \subseteq sm_i$.
- Immediacy. $\forall i, j : [(id_i, v_i) \in sm_j \ \wedge \ (id_j, v_j) \in sm_i] \ \Rightarrow \ (sm_i = sm_j)$.
- Termination. Any invocation of WS.write_snapshot() by a correct process terminates.

```
r_i ← 0;
loop forever r_i ← r_i + 1;
             local computations; compute v_i;
             sm_i ← WS[r_i].write_snapshot(v_i);
             local computations
end loop.
```

Figure 2: Generic algorithm for the iterated write-snapshot model (code for p_i)

The iterated model In the *iterated model* (IM) the shared memory is made up of an infinite number of one-shot write-snapshot objects $WS[1]$, $WS[2]$, ... These objects are accessed sequentially and asynchronously by each process, according to the classical round-based pattern described in Figure 2, where r_i denotes the current round number of process p_i. Although the IM restricts the order in which processes can access the shared objects, there is no limitation in terms of task solvability [5], even when the IM is enriched with certain objects more powerful than read/write registers [16].

When the IM is enriched with objects of type X, the system additionally has an infinite number of objects of type X, $X[1]$, $X[2]$, ..., which are accessed sequentially but asynchronously. Therefore, in round r, each process access $WR[r]$ and $X[r]$. Section 3.4 considers the IM enriched with GSB objects, more specifically perfect renaming and $(n+1)$-renaming objects.

3 From GSB tasks to (n, k)-set agreement

3.1 Two preliminary results

This section presents two results that will be used in Sections 3.2 and 3.4 for proving impossibility results from perfect renaming to set agreement.

Lemma 1 uses an idea that is recurrent in the rest of the paper: given an algorithm $\mathcal{A}$ for n processes that solves a task from a certain class of objects, one can slightly modify $\mathcal{A}$ to achieve an algorithm $\mathcal{B}$, possibly for less than n processes, that solves the same task from a distinct class of objects.

Lemma 1 *For* $1 \leq k \leq n$:
$\big(\langle n+1, n+1, 1, 1\rangle\text{-GSB} \rightarrow (n+1, k)\text{-SA}\big) \Rightarrow \big(\langle n, n+1, 0, 1\rangle\text{-GSB} \rightarrow (n, k)\text{-SA}\big)$.

Proof Let $\mathcal{A}$ be an algorithm that solves $(n+1, k)$-SA from $\langle n+1, n+1, 1, 1\rangle$-GSB. Consider the set of executions S of $\mathcal{A}$ in which only $p_1, \ldots, p_n$ participate. Observe that for any execution E of S, the collection of outputs that $p_1, \ldots, p_n$ receive in any invocation to an $\langle n+1, n+1, 1, 1\rangle$-GSB object, are valid outputs for $\langle n, n+1, 0, 1\rangle$-GSB.

We get a new algorithm $\mathcal{B}$ by modifying p_i's code, $1 \leq i \leq n$, as follows (p_{n+1} does not change). Each $\langle n+1, n+1, 1, 1\rangle$-GSB object of $\mathcal{A}$ is replaced by an $\langle n, n+1, 0, 1\rangle$-GSB object. Observe that for any execution E of $\mathcal{B}$ with participating set $p_1, \ldots, p_n$, there is an execution in S that is the same as E. Moreover, notice that $p_1, \ldots, p_n$ decide at most k distinct values in E. Therefore, suppressing p_{n+1} from $\mathcal{B}$, we obtain an algorithm $\mathcal{B}'$ for n processes that solves (n, k)-SA from $\langle n, n+1, 0, 1\rangle$-GSB. $\square_{Lemma\ 1}$

The following lemma is in some sense the opposite of Lemma 1. Roughly speaking, it says that given an impossibility result for $(n-1)$-slot, i.e., $\langle n, n-1, 1, n\rangle$-GSB, one can obtain an impossibility result for perfect renaming on $n' \geq n$ processes.

Lemma 2 *For* $1 \leq k \leq n-1$:
$\big(\langle n, n-1, 1, n\rangle\text{-GSB} \nrightarrow (n, k)\text{-SA}\big) \Rightarrow \big(\forall n' \geq n+1 :\ \langle n', n', 1, 1\rangle\text{-GSB} \nrightarrow (n', k)\text{-SA}\big)$.

Proof First, it is proved in [23] that $(n-1)$-slot on n processes can implement $(n+1)$-renaming on n-processes, that is

$$\langle n, n-1, 1, n\rangle\text{-GSB} \rightarrow \langle n, n+1, 0, 1\rangle\text{-GSB}. \tag{1}$$

By (1) and since by hypothesis $\langle n, n-1, 1, n\rangle$-GSB $\nrightarrow (n, k)$-SA, we get

$$\langle n, n+1, 0, 1\rangle\text{-GSB} \nrightarrow (n, k)\text{-SA}. \tag{2}$$

Also Lemma 1 and (2) imply

$$\langle n+1, n+1, 1, 1\rangle\text{-GSB} \nrightarrow (n+1, k)\text{-SA}. \tag{3}$$

Now, in [23] it is proved that perfect renaming on $n+1$ processes can implement any GSB task on $n+1$ processes, thus $\langle n+1, n+1, 1, 1\rangle$-GSB $\rightarrow \langle n+1, n+2, 0, 1\rangle$-GSB, and hence by (3)

$$\langle n+1, n+2, 0, 1\rangle\text{-GSB} \nrightarrow (n+1, k)\text{-SA}. \tag{4}$$

Therefore, by (4) and Lemma 1 on $n+2$ processes we get

$$\langle n+2, n+2, 1, 1\rangle\text{-GSB} \nrightarrow (n+2, k)\text{-SA}. \tag{5}$$

Let us repeat the same reasoning for $n+2$. Let us observe that
$\langle n+2, n+2, 1, 1\rangle$-GSB $\rightarrow \langle n+2, n+3, 0, 1\rangle$-GSB (proved in [23]). Hence, from this observation and (5) we have

$$\langle n+2, n+3, 0, 1\rangle\text{-GSB} \nrightarrow (n+2, k)\text{-SA}. \tag{6}$$

Thus, by (6) and Lemma 1 on $n+3$ processes

$$\langle n+2, n+2, 1, 1\rangle\text{-GSB} \nrightarrow (n+2, k)\text{-SA}. \tag{7}$$

Repeating the same for $n+3, n+4$, etc., the lemma follows. $\square_{Lemma\ 2}$

3.2 From perfect renaming to set agreement

This section shows a possibility and two impossibility results from perfect renaming to set agreement. The possibility result, Theorem 1, relies on the following lemma.

Lemma 3 *For* $n = 2 : \langle n, n, 1, 1\rangle$-GSB $\rightarrow (n, 1)$-T&S.

Proof Consider an object $\mathcal{O}$ that solves $\langle 2, 2, 1, 1\rangle$-GSB. By Theorem 2 of [23], we can assume that $\mathcal{O}$ is comparison-based. Consider an $i \in \{1, 2\}$ and let E be an execution of $\mathcal{O}$ in which only p_i participates (solo execution). It follows from the fact that $\mathcal{O}$ is comparison-based that p_i cannot use its input name in E. Consequently, the output name of p_i in this execution is not a function of its input name. Therefore in every solo execution, no matter its input name, p_i always decides the same output name, say v. Now, as p_i decides v in a solo execution and $\mathcal{O}$ is index-independent, we can conclude that in any solo execution in which p_j participates ($j \in \{1, 2\}$ and $j \neq i$), whatever its input name, p_j decides also v. Hence, the output name decided in a solo execution of $\mathcal{O}$ is predetermined. Let v be this output name.

We claim that we can solve $(2, 1)$-T&S from $\mathcal{O}$: each process calls $\mathcal{O}$ with its input name as input and decides 0 (winner) if it receives v (the value that $\mathcal{O}$ always outputs in a solo execution), otherwise it decides 1 (loser). The correctness proof of this implementation is the following. In a solo execution the participating processes obtains v from $\mathcal{O}$, and hence decides 0 (winner). In an execution in which the two processes of the system participate, exactly one of them obtain v from $\mathcal{O}$, by the specification of $\langle 2, 2, 1, 1\rangle$-GSB, and thus exactly one process decides 0 (winner) and exactly one process decides 1 (looser). The lemma follows. $\square_{Lemma\ 3}$

Theorem 1 *For* $n = 2 : \langle n, n, 1, 1\rangle$-GSB $\rightarrow (n, 1)$-SA.

Proof By Lemma 3, $\langle 2, 2, 1, 1\rangle$-GSB $\rightarrow (2, 1)$-T&S. Also it is known that $(2, 1)$-T&S $\rightarrow (2, 1)$-SA [19], and hence $\langle 2, 2, 1, 1\rangle$-GSB $\rightarrow$ $(2, 1)$-SA. $\square_{Theorem\ 1}$

Roughly speaking, the proofs of following two theorems consist on showing that $\langle n, n-1, 1, n\rangle$-GSB $\nrightarrow (n, k)$-SA for a small value of n, and then apply Lemma 2.

Theorem 2 $\forall n \geq 3 :$ $\langle n, n, 1, 1\rangle$-GSB $\nrightarrow (n, 1)$-SA.

Proof Consider the task $\langle 2, 1, 1, 2\rangle$-GSB. Obviously $\langle 2, 1, 1, 2\rangle$-GSB is read/write wait-free solvable and hence $\langle 2, 1, 1, 2\rangle$-GSB $\nrightarrow$ $(2, 1)$-SA, as $(2, 1)$-SA is not read/write solvable [11]. Therefore, by Lemma 2, $\forall n \geq 3$, $\langle n, n, 1, 1\rangle$-GSB $\nrightarrow (n, 1)$-SA. $\square_{Theorem\ 2}$

Theorem 3 $\forall n \geq 4 :$ $\langle n, n, 1, 1\rangle$-GSB $\nrightarrow (n, 2)$-SA.

Proof Observe that $\langle 3, 2, 1, 3\rangle$-GSB is WSB on 3 processes, hence equal to $(2n-2)$-renaming on 3 processes, by Corollary 2 of [17]. Also in [17] it is proved that $(n, n-1)$-SA is strictly stronger than $(2n-2)$-renaming for odd n, hence $\langle 3, 2, 1, 3\rangle$-GSB $\nrightarrow (3, 2)$-SA. Therefore, by Lemma 2, $\forall n \geq 4$, $\langle n, n, 1, 1\rangle$-GSB $\nrightarrow (n, 2)$-SA. $\square_{Theorem\ 3}$

3.3 From $(2n-2)$-renaming to set agreement

As mentioned in the Introduction, it is proved in [17] that $\langle n, 2n-2, 0, 1\rangle$-GSB $\nrightarrow (n, n-1)$-SA, for odd n (recall that $\langle n, 2n-2, 0, 1\rangle$-GSB is $(2n-2)$-renaming on n processes). Note that for $k = n-1$, $\langle n, 2n-2, 0, 1\rangle$-GSB is $\langle n, 2n - \lceil \frac{n}{k} \rceil, 0, 1\rangle$-GSB.

Now, it is proved in [7] that if $\gcd(\binom{n}{1}, \ldots, \binom{n}{n-1}) = 1$ (and only in this case), then $\langle n, 2n-2, 0, 1\rangle$-GSB is read/write wait-free solvable. Thus, for these values of n, $\langle n, 2n-2, 0, 1\rangle$-GSB $\nrightarrow (n, n-1)$-SA, since it has been proved [3, 22, 26] that $(n, n-1)$-SA is not read/write wait-free solvable. There are odd and even values of n holding this property. What does it happen when n is even and $\gcd(\binom{n}{1}, \ldots, \binom{n}{n-1}) \neq 1$?

It is observed in [6] that if $\gcd(\binom{n}{1}, \ldots, \binom{n}{n-1}) \neq 1$ then n is a prime power. The only case in which n is prime power and even is $n = 2^x, x \geq 1$. However, for the case $n = 2$, $\langle n, 2n-2, 0, 1\rangle$-GSB is equivalent to perfect renaming, and hence cannot implement $(n, n-1)$-SA, by Theorem 2. All these results partially answer the question left open in [17]. Observe that the result in [17] is extended for infinitely many values of n.

Theorem 4 *For $n = 2$, $\langle n, 2n-2, 0, 1\rangle$-GSB $\leftrightarrow (n, n-1)$-SA, $\forall n$, $(n, n-1)$-SA $\rightarrow \langle n, 2n-2, 0, 1\rangle$-GSB, and $\forall n \neq 2^x, x > 1$, $\langle n, 2n-2, 0, 1\rangle$-GSB $\nrightarrow (n, n-1)$-SA.*

Theorem 5 below shows that for any n, $\langle n, 2n-2, 0, 1\rangle$-GSB is not strong enough for solving $(n, n-2)$-SA. Of course all cases $n \neq 2^x$, $x > 1$, are implied by Theorem 4, however, the proof of Theorem 5 is simple (uses the same idea as the one used in the proof of Lemma 1) and self-contained.

Lemma 4 $\forall n$: $\big(\langle n, 2n-2, 0, 1\rangle\text{-GSB} \rightarrow (n, n-2)\text{-SA}\big) \Rightarrow \big(\langle n-1, 2n-2, 0, 1\rangle\text{-GSB} \rightarrow (n-1, n-2)\text{-SA}\big)$.

Proof Let $\mathcal{A}$ be an algorithm that solves $(n, n-2)$-SA from $\langle n, 2n-2, 0, 1\rangle$-GSB. Consider the set of executions S of $\mathcal{A}$ in which only $p_1, \ldots, p_{n-1}$ participate. Observe that for any execution E of S, the collection of outputs that $p_1, \ldots, p_{n-1}$ receive in any invocation to an $\langle n, 2n-2, 0, 1\rangle$-GSB object, are valid outputs for $\langle n-1, 2n-2, 0, 1\rangle$-GSB. We get a new algorithm $\mathcal{B}$ by modifying p_i's code, $1 \leq i \leq n-1$, as follows. Each $\langle n, 2n-2, 0, 1\rangle$-GSB object of $\mathcal{A}$ is replaced by an $\langle n-1, 2n-2, 0, 1\rangle$-GSB object. Observe that for any execution E of $\mathcal{B}$ with participating set $p_1, \ldots, p_{n-1}$, there is an execution in S that is the same as E. Moreover, at most $n-2$ distinct values are decided in E. Therefore, suppressing p_n from $\mathcal{B}$, we obtain an algorithm $\mathcal{B}'$ for $n-1$ processes that solves $(n-1, n-2)$-SA from $\langle n-1, 2n-2, 0, 1\rangle$-GSB. $\square_{Lemma\ 4}$

Theorem 5 $\forall n$, $\langle n, 2n-2, 0, 1\rangle$-GSB $\nrightarrow (n, n-2)$-SA.

Proof Suppose, for the sake of contradiction, that $\langle n, 2n-2, 0, 1\rangle$-GSB $\rightarrow (n, n-2)$-SA. Hence, $\langle n-1, 2n-2, 0, 1\rangle$-GSB $\rightarrow (n-1, n-2)$-SA, by Lemma 4. In [1] it is proved that M-renaming on $n-1$ processes, i.e., $\langle n-1, M, 0, 1\rangle$-GSB, is read/write wait-free solvable if $M \geq 2(n-1) - 1 = 2n-3$. Therefore, $\langle n-1, 2n-2, 0, 1\rangle$-GSB is read/write wait-free solvable, from which follows that $(n-1, n-2)$-SA is read/write wait-free solvable, since $\langle n-1, 2n-2, 0, 1\rangle$-GSB $\rightarrow (n-1, n-2)$-SA. However, it has been proved that $(n-1, n-2)$-SA is not read/write wait-free solvable [3, 22, 26]. A contradiction. $\square_{Theorem\ 5}$

3.4 From perfect renaming to set agreement in the iterated model

This section proves Theorem 7 below, which states that in IM, $\langle n, n, 1, 1\rangle$-GSB $\nrightarrow (n, n-2)$-SA. For the rest of the section we only consider IM. The strategy of the proof is the following.

1. Given the task $\langle n, n+1, 0, 1\rangle$-GSB, we define a new n-process task T such that if $\langle n, n+1, 0, 1\rangle$-GSB $\rightarrow (n, n-1)$-SA, then $T \rightarrow (n, n-1)$-SA.
2. Using a result in [17], we will prove that $T \nrightarrow (n, n-1)$-SA, hence $\langle n, n+1, 0, 1\rangle$-GSB $\nrightarrow (n, n-1)$-SA, by the previous item.
3. Finally, the fact that $\langle n, n+1, 0, 1\rangle$-GSB $\nrightarrow (n, n-1)$-SA and a simple observation that Lemma 1 holds in IM, implies that $\langle n+1, n+1, 1, 1\rangle$-GSB $\nrightarrow (n+1, n-1)$-SA, from which follows the desired result.

For proving Theorem 7, we model tasks using notions from combinatorial topology as in many papers (see for example [2, 3, 17, 22, 26]). We review some basic concepts.

A *simplex* σ is a finite set. The elements of a simplex are its *vertexes*. The dimension of a simplex σ is the number of its vertexes minus 1. If σ has $n+1$ vertexes then it is called an *n-simplex*. A simplex τ is a *face* of σ if τ is a subset of σ. If τ is not equal to σ then τ is a *proper face* of σ. A *complex* $\mathcal{K}$ is a set of simplexes, closed under containment. The dimension of a complex $\mathcal{K}$ is the maximum dimension of its simplexes. A complex $\mathcal{K}$ of dimension n is an *n-complex*. For a simplex σ, we often denote as σ the complex containing all faces of σ (including σ itself). A complex $\mathcal{L}$ is a *subcomplex* of the complex $\mathcal{K}$ if $\mathcal{L} \subseteq \mathcal{K}$.

An n-complex is *connected* if any two n-simplexes can be joined by a sequence of n-simplexes in which each pair of neighboring simplexes share an $(n-1)$-face. An n-complex $\mathcal{K}$ is an *n-manifold* (sometimes called *pseudo-manifold*) if it is connected and each of its $(n-1)$-simplexes is contained in one or two n-simplexes. We say that an $(n-1)$-simplex of $\mathcal{K}$ is *internal* if it is contained in two n-simplexes, otherwise it is *external*. The *boundary* of an n-manifold $\mathcal{K}$, denoted $\partial\mathcal{K}$, is the subcomplex induced by its external $(n-1)$-simplexes. For example, the complex induced by an n-simplex σ and all its faces is a n-manifold with boundary; its boundary $\partial\sigma$ contains all $(n-1)$-faces of σ.

For a domain of inputs I, the *input complex* $\mathcal{I}$ is an $(n-1)$-complex that contains $(n-1)$-simplexes (subsets with n elements) of $\{1,\ldots,n\} \times I$, and all their faces, such that no pair of vertexes have the same index, the first entry of each pair. An *output complex*, $\mathcal{O}$, over a domain of outputs O, is defined similarly. The meaning of a vertex (i,v) of $\mathcal{I}$ (resp. $\mathcal{O}$) is that process with index i has input (resp. output) v.

A *task* is defined as a triple $\langle \mathcal{I}, \mathcal{O}, \triangle \rangle$, where $\mathcal{I}$ is an input complex, $\mathcal{O}$ is an output complex and $\triangle$ is a recursive map carrying each m-simplex σ of $\mathcal{I}$, $0 \leq m \leq n-1$, to a non-empty m-subcomplex of $\mathcal{O}$. This definition has the following operational interpretation: $\triangle(\sigma)$ is the set of legal final states in executions where only the $m+1$ processes in σ participate (the rest fail without taking any steps).

The concept of *manifold task* is introduced in [17]. It is proved there that manifold tasks are too weak to solve $(n, n-1)$-SA. Manifold tasks are used in [17] for proving that in IM, for odd n, $\langle n, 2n-2, 0, 1\rangle$ is strictly weaker than $(n, n-1)$-SA (the result is extended for the general model in [16]).

Definition 1 ([17]) *A task $\langle \mathcal{I}, \mathcal{O}, \triangle \rangle$ is a* manifold task *if for every m-simplex $\sigma \in \mathcal{I}$, $0 \leq m \leq n-1$, $\triangle(\sigma)$ is a m-manifold and $\partial\triangle(\sigma) = \triangle(\partial\sigma)$.*

Lemma 5 ([17]) *No manifold task can solve $(n, n-1)$-SA.*

We are interested in the task $\langle n, n+1, 0, 1\rangle$-GSB. This task is modeled as $\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R \rangle$, where:

- The domain of inputs is $\{1,\ldots,N\}$ and no pair of vertexes of a simplex in $\mathcal{I}_R$ have the same input value.
- The outputs are $\{1,\ldots,n+1\}$ and no pair of vertices of a simplex in $\mathcal{O}_R$ have the same output value.
- For each m-simplex $\sigma \in \mathcal{I}_R$, $\triangle_R(\sigma)$ is the m-subcomplex of $\mathcal{O}_R$ containing every m-simplex that has the same indexes at its vertexes as σ. Therefore, for each $(n-1)$-simplex $\sigma \in \mathcal{I}_R$, $\triangle_R(\sigma) = \mathcal{O}_R$.

Using $\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R \rangle$ we define a task T, whose main properties are that (1) it solves (n,k)-SA, provided that $\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R \rangle$ solves (n,k)-SA, and (2) it is a manifold task. The task T is defined as $\langle \mathcal{I}_T, \mathcal{O}_T, \triangle_T \rangle$, with $\mathcal{I}_T = \mathcal{I}_R$, $\mathcal{O}_T = \mathcal{O}_R$, and $\triangle_T$ is defined as follows:

- For $0 \leq m \leq n-2$, for each m-simplex σ of $\mathcal{I}$, $\triangle_T(\sigma)$ is the complex induced by the m-face of τ that has the same indexes at its vertexes as σ.
- For $m = n-1$, for or each $(n-1)$-simplex σ of $\mathcal{I}$, $\triangle_T(\sigma) = \mathcal{O}_T \setminus \tau$, where τ is the $(n-1)$-simplex $\{(1,1),\ldots,(n,n)\}$ of $\mathcal{O}_I$. Figure 3 depicts an example of complex $\mathcal{O}_T \setminus \tau$ of dimension 2.

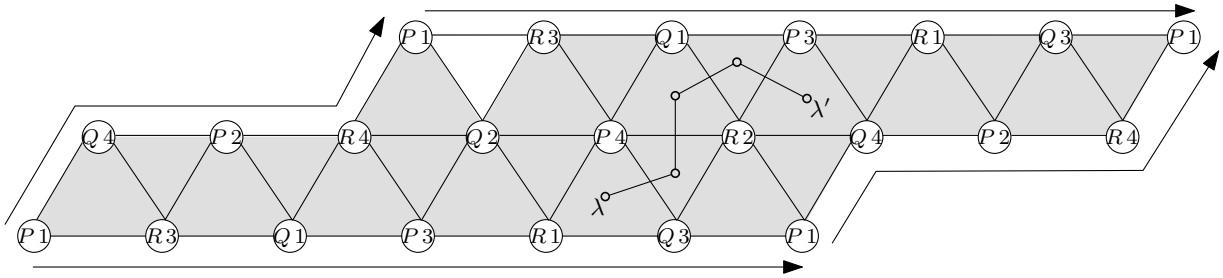

Figure 3: Complex $\mathcal{O}_T \setminus \tau$ of dimension 2 and a sequence produced by findSequence()

The proof of Lemma 6 uses a similar idea than the one used in the proof of Lemma 1.

Lemma 6 *For $1 \leq k \leq n-1$: $\big(\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R \rangle \rightarrow (n,k)\text{-SA}\big) \Rightarrow \big(\langle \mathcal{I}_T, \mathcal{O}_T, \triangle_T \rangle \rightarrow (n,k)\text{-SA}\big)$.*

Proof First, recall that $\mathcal{I}_R = \mathcal{I}_T$ and $\mathcal{O}_R = \mathcal{O}_T$. It is not hard to see that by the definition of $\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R \rangle$ and $\langle \mathcal{I}_T, \mathcal{O}_T, \triangle_T \rangle$, for every m-simplex $\sigma \in \mathcal{I}_R$,

$$\triangle_T(\sigma) \subset \triangle_R(\sigma), \tag{8}$$

for each $0 \leq m \leq n-1$. Therefore, any valid collection of outputs for $\langle \mathcal{I}_T, \mathcal{O}_T, \triangle_T \rangle$ is a valid collection of outputs for $\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R \rangle$.

Now, let $\mathcal{A}$ be an algorithm that solves (n,k)-SA from $\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R \rangle$. We get a new algorithm $\mathcal{B}$ by modifying p_i's code, $1 \leq i \leq n$, as follows. Each $\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R \rangle$ object in $\mathcal{A}$ is replaced by an $\langle \mathcal{I}_T, \mathcal{O}_T, \triangle_T \rangle$ object. As explained above, (8) says that any valid collection of outputs for $\langle \mathcal{I}_T, \mathcal{O}_T, \triangle_T \rangle$ is a valid collection of outputs for $\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R \rangle$. Therefore, for each execution E of $\mathcal{B}$, there is an execution of $\mathcal{A}$ that is the same as E, thus the participating processes of E decide at most k distinct values, i.e., $\langle \mathcal{I}_T, \mathcal{O}_T, \triangle_T \rangle \rightarrow (n,k)$-SA. $\square_{Lemma\ 6}$

Lemma 7 *The task $\langle \mathcal{I}_T, \mathcal{O}_T, \triangle_T \rangle$ is a manifold task.*

Proof Let τ denote the $(n-1)$-simplex $\{(1,1),\ldots,(n,n)\}$ of $\mathcal{O}_T$. First, for each $0 \leq m \leq n-2$, for every m-simplex $\sigma \in \mathcal{I}_T$, $\triangle_T(\sigma) = \tau'$, where τ' is the m-face of τ that has the same indexes at its vertexes as σ. Clearly $\triangle_T(\sigma)$ is an m-manifold with $\partial\triangle_T(\sigma) = \partial\tau' = \triangle(\partial\sigma)$.

Now consider an $(n-1)$-simplex $\sigma \in \mathcal{I}_T$. We have that $\triangle_T(\sigma) = \mathcal{O}_T \setminus \tau$. We will first prove that $\mathcal{O}_T$ is an $(n-1)$-manifold without boundary, and then show that this implies that $\mathcal{O}_T \setminus \tau$ is a $(n-1)$-manifold (with boundary) and $\partial\triangle_T(\sigma) = \triangle(\partial\sigma)$.

For proving that $\mathcal{O}_T$ is an $(n-1)$-manifold without boundary, we prove the two following claims: (C1) Each $(n-2)$-simplex of $\mathcal{O}_T$ is contained in exactly two $(n-1)$-simplexes, and (C2) $\mathcal{O}_T$ is connected. Let us first prove claim C1. Consider an $(n-2)$-simplex λ of $\mathcal{O}_T$. By definition of $\mathcal{O}_T$, λ has the form $\{(i_1, nm_1), \ldots, (i_{n-1}, nm_{n-1})\}$, where for all $1 \leq j_1 \neq j_2, k_1 \neq k_2 \leq n-1$, we have $i_{j_1} \neq i_{j_2}$ and $nm_{k_1} \neq nm_{k_2}$. Note that there is exactly one index process in $\{1,\ldots,n\}$ that does not appear in λ, and similarly there are exactly two output names in $\{1,\ldots,n+1\}$ that do not appear in λ. Let j be such index and k_1 and k_2 be such output names. By definition of $\mathcal{O}_T$, $(n-1)$-simplexes $\gamma_1 = \lambda \cup \{(j,k_1)\}$ and $\gamma_2 = \lambda \cup \{(j,k_2)\}$ belong to $\mathcal{O}_T$. Moreover, clearly λ is contained in γ_1 and γ_1, and also there is no other $(n-1)$-simplex of $\mathcal{O}_T$ containing λ because $\mathcal{O}_T$ is defined over the output space $\{1,\ldots,n+1\}$.

Let us now prove claim C2, namely, $\mathcal{O}_T$ is connected. Given two $(n-1)$-simplexes $\lambda, \lambda' \in \mathcal{O}_T$, the sequential algorithm findSequence() in Figure 4 outputs a sequence S of $(n-1)$-simplexes of $\mathcal{O}_T$ such that each pair of neighboring simplexes share an $(n-2)$-face, and the first and last simplexes are λ and λ'. Without loss of generality, the algorithm assumes that λ and λ' have the form $\{(i_1, nm_i), \ldots, (i_n, nm_n)\}$ and $\{(i_1, nm'_i), \ldots, (i_n, nm'_n)\}$, respectively. Algorithm findSequence() uses the following notation. Let γ be an $(n-1)$ simplex of $\mathcal{O}_T$. Consider an index $i \in \{1,\ldots,n\}$ and let (j,x) be the vertex of γ such that $i = j$. The $(n-2)$-face $\gamma - \{(j,x)\}$ of γ is denoted as $\gamma|i$. This notation is used in lines 05 and 10 of findSequence(). Consider and an output name $y \in \{1,\ldots,n+1\}$ such that there is no $(j,x) \in \gamma$ such that $y = x$. The $(n-2)$-face $\gamma - \{(j,x)\}$ of γ is denoted as $\gamma|x$. This notation is used in line 09 of findSequence(). The dot $(\cdot)$ operator in lines 06, 11 and 14 means concatenation. Figure 3 contains an example of a path produced by findSequence().

For the j-th iteration of the loop in line 02, $1 \leq j \leq n$, let γ^j be the last simplex of the sequence S at the beginning of the iteration. An invariant (I) of findSequence() is that at the beginning of the j-th iteration, S is a sequence of $(n-1)$-simplexes of $\mathcal{O}_T$ such that each pair of neighboring simplexes share an $(n-2)$-face, and also for every vertex $(i,x) \in \gamma^j$, if $i = i_k$ for some $1 \leq k \leq j-1$, then $x = nm'_k$ (recall that $\lambda = \{(i_1, nm_i), \ldots, (i_n, nm_n)\}$ and $\lambda' = \{(i_1, nm'_i), \ldots, (i_n, nm'_n)\}$). Observe that invariant (I) implies that, at line 14, the last simplex γ of S shares an $(n-2)$-face with λ'. In particular, γ and λ' share $\gamma|i_n$; in fact it can be that $\gamma = \lambda'$. Therefore, $S \cdot \lambda'$ in line 14 is the desired sequence of simplexes. Figure 3 contains an example of a path produced by findSequence().

By induction we prove invariant (I). For $j = 1$, it clearly holds, since $S = \lambda$. Suppose that (I) holds for $1 \leq j \leq n-1$; we will prove it holds for $j+1$. Consider the simplex γ in line 03 in the j-th iteration of findSequence(). By assumption, for every vertex $(i,x) \in \gamma$, if $i = i_k$ for some $1 \leq k \leq j-1$, then $x = nm'_k$. We have two cases. If nm'_j does not appear in γ, i.e., line 04 is false, then γ' in line 05 and γ share the $(n-2)$-face $\gamma|i_j$, since $\gamma' = \gamma|i_j \cup \{(i_j, nm'_j)\}$. Moreover, observe that for every vertex $(i,x) \in \gamma'$, if $i = i_k$ for some $1 \leq k \leq j$, then $x = nm'_k$. In addition, $\gamma' \in \mathcal{O}_T$ because, by induction hypothesis, $\gamma|i_j \in \mathcal{O}_T$ and $nm' \in \{1,\ldots,n+1\}$. Therefore, invariant (I) holds for $j+1$.

In the second case we have that nm'_j indeed appears in γ, that is line 04 is true. Thus γ' in line 09 and γ share the $(n-2)$-face $\gamma|nm'_j$, because $\gamma' = \gamma|nm'_j \cup \{(i,x)\}$. Also, $\gamma' \in \mathcal{O}_T$, by the definition of i and x, and because $\gamma|nm'_j \in \mathcal{O}_T$, by induction hypothesis. Moreover, since for all $1 \leq k_1 \neq k_2 \leq n$, $nm'_{k1} \neq nm'_{k2}$, it cannot be that $i = i_k$, for some $1 \leq k \leq j-1$, which implies that for every vertex $(\ell,x) \in \gamma'$, if $\ell = i_k$ for some $1 \leq k \leq j-1$, then $x = nm'_k$. Observe that we now have the same situation as in the previous case, considering the sequence $S \cdot \gamma'$, thus the same argument used for proving that case shows that γ' and γ'' in line 10 share an $(n-2)$-face, and for every vertex $(\ell,x) \in \gamma''$, if $\ell = i_k$ for some $1 \leq k \leq j$, then $x = nm'_k$. Invariant (I) holds for $j+1$.

Until here we have proved that $\mathcal{O}_T$ is a manifold without boundary. We now prove that $\mathcal{O}_T \setminus \tau$ is a manifold with boundary, where τ is the $(n-1)$-simplex $\{(1,1),\ldots,(n,n)\}$ of $\mathcal{O}_T$. Moreover, we prove that $\partial(\mathcal{O}_T \setminus \tau) = \partial\tau$. The fact that $\mathcal{O}_T \setminus \tau$ is connected comes from the observation that the correctness of algorithm findSequence() does not depend at all on the order $i_1,\ldots,i_n$ of the index processes (recall that, for findSequence(), $\lambda = \{(i_1, nm_i), \ldots, (i_n, nm_n)\}$ and $\lambda' = \{(i_1, nm'_i), \ldots, (i_n, nm'_n)\}$). Thus, if during the execution of findSequence(), appears τ in the sequence S in some iteration, say in the j-th iteration, it is enough to restart the execution with $\lambda = \{(i_1, nm_i), \ldots, (i_{j+1}, nm_{j+1}), (i_j, nm_j), \ldots, (i_n, nm_n)\}$ and $\lambda = \{(i_1, nm'_i), \ldots, (i_{j+1}, nm'_{j+1}), (i_j, nm'_j), \ldots, (i_n, nm'_n)\}$, i.e., swapping i_j with i_{j+1}. This "evades" the simplex τ. Now, because $\mathcal{O}_T$ is a manifold without boundary, each $(n-2)$-simplex of $\mathcal{O}_T \setminus \tau$ is contained in one or two $(n-1)$-simplexes. Moreover, it is not hard to see that an $(n-2)$-simplex of $\mathcal{O}_T \setminus \tau$ is contained in exactly one $(n-1)$-simplex if and only if it is face of τ, namely, it belongs to $\partial\tau$. Thus, $\mathcal{O}_T \setminus \tau$ is a manifold with $\partial(\mathcal{O}_T \setminus \tau) = \partial\tau$.

```
operation findSequence(λ, λ'):
(01) sequence S ← λ;
(02) for j from 1 to n − 1 do
(03)     γ ← last element of S;
(04)     if (∄(k, y) ∈ γ s.t. nm'_j = y)
(05)         then γ' ← (γ|i_j) ∪ {(i_j, nm'_j)};
(06)              S ← S · γ';
(07)         else let x be the output name of {1, ..., n + 1} s. t. ∄(k, y) ∈ γ s. t. x = y;
(08)              let i be the index of {1, ..., n} s. t. ∄(k, y) ∈ γ|nm'_j s. t. i = k;
(09)              γ' ← (γ|nm'_j) ∪ {(i, x)};
(10)              γ'' ← (γ'|i_j) ∪ {(i_j, nm'_j)};
(11)              S ← S · γ' · γ'';
(12)     end if
(13) end for
(14) return(S · λ');
```

Figure 4: Finding a sequence of simplexes that joins two $(n-1)$-simplexes $\lambda, \lambda' \in \mathcal{O}_T$

Consider an $(n-1)$-simplex $\sigma \in \mathcal{I}_T$. We have that $\triangle_T(\sigma) = \mathcal{O}_T \setminus \tau$ is an $(n-1)$-manifold. In addition, $\partial\triangle_T(\sigma) = \partial\tau = \triangle(\partial\sigma)$, since $\partial(\mathcal{O}_T \setminus \tau) = \partial\tau$ and because by definition of $\triangle_T$, for each $(n-2)$-simplex of $\partial\sigma$, $\triangle_T(\sigma)$ is the complex induced by the m-face of τ that has the same indexes at its vertexes as σ. The lemma follows. $\square_{Lemma\ 7}$

We now are ready to prove the main results of this section.

Theorem 6 *In* IM, $\forall n : \langle n, n+1, 0, 1\rangle$-GSB $\nrightarrow (n, n-1)$-SA.

Proof By Lemmas 5 and 7, we get $\langle \mathcal{I}_T, \mathcal{O}_T, \triangle_T\rangle \nrightarrow (n, n-1)$-SA, and by Lemma 6, $\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R\rangle \nrightarrow (n, n-1)$-SA. Thus $\langle n, n+1, 0, 1\rangle$-GSB $\nrightarrow (n, n-1)$-SA, since $\langle n, n+1, 0, 1\rangle$-GSB is $\langle \mathcal{I}_R, \mathcal{O}_R, \triangle_R\rangle$. $\square_{Theorem\ 6}$

Theorem 7 *In* IM, $\forall n : \langle n, n, 1, 1\rangle$-GSB $\nrightarrow (n, n-2)$-SA.

Proof It is straightforward to see that Lemma 1 holds in IM; the proof is essentially the same. Now, by Theorem 6 we have that $\forall n : \langle n, n+1, 0, 1\rangle$-GSB $\nrightarrow (n, n-1)$-SA, and from Lemma 1 with $k = n-2$, it follows that $\forall n : \langle n+1, n+1, 0, 1\rangle$-GSB $\nrightarrow (n+1, n-1)$-SA. Therefore, in the end we get $\forall n : \langle n, n, 1, 1\rangle$-GSB $\nrightarrow (n, n-2)$-SA. $\square_{Theorem\ 7}$

4 From (n, k)-set agreement to GSB

4.1 From set agreement to perfect renaming

Lemma 8 uses the same idea as the one used in Lemma 1 in Section 3.1. Theorems 8 and 9 below are proved using Lemma 8.

Lemma 8 *For* $1 \leq k \leq n-1 : \big((n, k)$-SA $\rightarrow \langle n, n, 1, 1\rangle$-GSB$\big) \Rightarrow \big(\langle k, n, 0, 1\rangle$-GSB *is read/write wait-free solvable*$\big)$.

Proof Let $\mathcal{A}$ be an algorithm that implements $\langle n, n, 1, 1\rangle$-GSB from (n, k)-SA. Consider the set of executions S of $\mathcal{A}$ in which only $p_1, \ldots, p_k$ participate. Observe that for any execution E of S, the outputs of $p_1, \ldots, p_k$ in E are valid outputs for $\langle k, n, 0, 1\rangle$-GSB. Note that S contains a set S' of executions in which each invocation by p_i, $1 \leq i \leq k$, to any (n, k)-SA object outputs the value p_i uses as input.

Let $\mathcal{B}$ be the algorithm obtained by modifying p_i's code in $\mathcal{A}$, $1 \leq i \leq k$, as follows (the codes of p_{k+1} until p_n are not modified). Each invocation to an (n, k)-SA object is replaced by a function that just outputs the input it receives. For any execution E of $\mathcal{B}$ with participating set $p_1, \ldots, p_k$, there is an execution in S' that is the same as E. Then, suppressing the processes $p_{k+1}, \ldots, p_n$ from $\mathcal{B}$, we obtain an algorithm $\mathcal{B}'$ for k processes that read/write solves $\langle k, n, 0, 1\rangle$-GSB. $\square_{Lemma\ 8}$

Theorem 8 $\forall n > 2$*:* $(n, n-1)$-SA $\nrightarrow \langle n, n, 1, 1\rangle$-GSB.

Proof Suppose, for the sake of contradiction, that $(n, n-1)$-SA $\rightarrow \langle n, n, 1, 1\rangle$-GSB. It then follows from Lemma 8 with $k = n-1$ that $\langle n-1, n, 0, 1\rangle$-GSB is read/write wait-free solvable. However, it is proved in [1] that $\langle n-1, n, 0, 1\rangle$-GSB ($n$-renaming on $n-1$ processes) is not read/write wait-free solvable for any value of n. A contradiction. $\square_{Theorem\ 8}$

Theorem 9 *For* $n, k : \big(k \geq \lceil \frac{n}{2} \rceil + 1 \wedge \gcd(\binom{k}{1}, \ldots, \binom{k}{k-1}) \neq 1\big) \Rightarrow \big((n,k)\text{-SA} \nrightarrow \langle n, n, 1, 1\rangle\text{-GSB}\big)$.

Proof Suppose, for the sake of contradiction, that (n,k)-SA $\rightarrow \langle n, n, 1, 1\rangle$-GSB. It follows from Lemma 8 that $\langle k, n, 0, 1\rangle$-GSB is read/write wait-free solvable. Let us now observe that, as $2k-2 \geq n+(n \mod 2)$, it follows that $\langle k, n, 0, 1\rangle$-GSB $\rightarrow \langle k, 2k-2, 0, 1\rangle$-GSB. Moreover, it is proved in [7] that $\langle k, 2k-2, 0, 1\rangle$-GSB is not read/write solvable if $\gcd(\binom{k}{1}, \ldots, \binom{k}{k-1}) \neq 1$, from which the lemma follows. $\square_{Theorem\ 9}$

This section ends with the following simple result.

Theorem 10 $\forall n : (n,1)\text{-SA} \rightarrow \langle n, n, 1, 1\rangle$-GSB.

Proof The algorithm is very simple. The processes have an n-array A of $(n,1)$-SA objects. Starting from $i = 1$, each process invokes $A[i]$ with its input name as input; if it receives its input name from $A[i]$, it decides output name i, otherwise increases i by one and repeats. $\square_{Theorem\ 10}$

4.2 From (n,k)-set agreement to $\lceil \frac{n}{k} \rceil$-slot, $(n+k-1)$-renaming and $(2n - \lceil \frac{n}{k} \rceil)$-renaming

It is proved in [12] that $(n+k-1)$-renaming can be solved from (n,k)-set agreement, thus:

Theorem 11 $\forall n, k : (n,k)\text{-SA} \rightarrow \langle n, n+k-1, 0, 1\rangle$-GSB.

An algorithm is presented in [24] that solves n-process adaptive $(2p - \lceil \frac{p}{k} \rceil)$-renaming from (n,k)-SA. This algorithm implies the following.

Theorem 12 $\forall n, k : (n,k)\text{-SA} \rightarrow \langle n, 2n - \lceil \frac{n}{k} \rceil, 0, 1\rangle$-GSB.

In what follows we prove that (n,k)-SA $\rightarrow \langle n, \lceil \frac{n}{k} \rceil, 1, n\rangle$-GSB. Since (n,k)-SA solves (n,k)-participating set [24], it is enough to prove that (n,k)-participating set solves $\langle n, \lceil \frac{n}{k} \rceil, 1, n\rangle$-GSB. The (n,k)-*participating set* (PS) task is the one-shot write snapshot task (defined by self-inclusion, containment, immediacy and termination properties stated in Section 2.3) plus the following property (at most k processes output the same set):

- Bounded simultaneity. $\forall \ell, 1 \leq \ell \leq n : \ |\{j : |sm_j| = \ell\}| \leq k$.

Using an (n,k)-PS object we can solve $\langle n, \lceil \frac{n}{k} \rceil, 1, n\rangle$-GSB, as Lemma 9 shows.

Lemma 9 $\forall n, k : (n,k)\text{-PS} \rightarrow \langle n, \lceil \frac{n}{k} \rceil, 1, n\rangle$-GSB.

Proof It is easy to solve $\langle n, \lceil \frac{n}{k} \rceil, 1, n\rangle$-GSB from (n,k)-PS: every process p_i invokes the same (n,k)-PS object and decides $\lceil \frac{|sm_i|}{k} \rceil$, where sm_i is the set it receives from the (n,k)-PS object.

To prove the correctness of the algorithm we have to show that in every execution each one of the $\lceil \frac{n}{k} \rceil$ slots is decided by at least one process. Suppose, for the sake of contradiction, that there is an execution E in which no process decides some slot x, $1 \leq x \leq \lceil \frac{n}{k} \rceil$. The (n,k)-PS object outputs sets ordered by containment: $S_1 \subset \ldots \subset S_m$. Moreover, the processes p_i that get $sm_i = S_j$ are the processes p_i such that $(id_i, -) \in S_j \setminus S_{j-1}$ [4] (for $j = 0$, $S_{-1} = \emptyset$). Since no process decides x in E, it must be that for some j, $1 \leq j \leq m-1$, $|S_j| \leq (x-1)k$ and $|S_{j+1}| \geq xk + 1$ (otherwise the processes that get S_j or S_{j+1} would decide x). Observe that $|S_{j+1} \setminus S_j| \geq k+1$, hence at least $k+1$ processes received S_{j+1} from the (n,k)-PS object. A contradiction with the bounded simultaneity property. $\square_{Lemma\ 9}$

As mentioned above, it is proved in [24] that (n,k)-SA $\rightarrow (n,k)$-PS, thus the previous lemma implies the following result.

Theorem 13 $\forall n, k : (n,k)\text{-SA} \rightarrow \langle n, \lceil \frac{n}{k} \rceil, 1, n\rangle$-GSB.

Theorems 14 and 15 shows the relation between $\lceil \frac{n}{k} \rceil$-slot, $(n+k-1)$-renaming and $(2n - \lceil \frac{n}{k} \rceil)$-renaming.

Theorem 14 $\forall n, k : \langle n, n+k-1, 0, 1\rangle\text{-GSB} \rightarrow \langle n, 2n - \lceil \frac{n}{k} \rceil, 0, 1\rangle$-GSB.

Proof Let us observe that $n + k - 1 \leq 2n - \lceil \frac{n}{k} \rceil$, for $1 \leq k \leq n-1$. Therefore, an algorithm that solves $\langle n, \lceil \frac{n}{k} \rceil, 1, n\rangle$-GSB, also solves $\langle n, 2n - \lceil \frac{n}{k} \rceil, 0, 1\rangle$-GSB. Consequently, $\langle n, n+k-1, 0, 1\rangle$-GSB $\rightarrow$
$\langle n, 2n - \lceil \frac{n}{k} \rceil, 0, 1\rangle$-GSB. $\square_{Theorem\ 14}$

Theorem 15 $\forall n, k : \langle n, n+k-1, 0, 1\rangle\text{-GSB} \rightarrow \langle n, \lceil \frac{n}{k} \rceil, 1, n\rangle$-GSB.

Proof Consider the following algorithm $\mathcal{A}$. Each process first calls an $\langle n, n+k-1, 0, 1\rangle$-GSB object X and then decides $(y \mod \lceil \frac{n}{k} \rceil) + 1$, where y is the value it receives from X. In what follows we prove that $\mathcal{A}$ solves $\langle n, \lceil \frac{n}{k} \rceil, 1, n\rangle$-GSB, i.e., in every execution, each element of $\{1, \ldots, \lceil \frac{n}{k} \rceil\}$ is decided by at least one process.

For the rest of the proof let α be $\lceil \frac{n}{k} \rceil$. First observe that mod operator splits the integers $1, \ldots, n+k-1$ into α equivalence classes, $[0], \ldots, [\alpha-1]$. Let $\#[j]$ denotes the number of elements of the equivalence class $[j]$. Each one of these equivalence classes contains roughly the same amount of elements, i.e., $\forall i, j : |\#[i] - \#[j]| \leq 1$. We will prove the following inequality, which, as explained in detail below, implies the correctness of $\mathcal{A}$. For every $i \in \{0, \ldots, \alpha-1\}$,

$$\sum_{j=0, j\neq i}^{\alpha-1} \#[j] < n. \tag{9}$$

We prove inequality (9). Let β be $\lfloor \frac{n+k-1}{\alpha} \rfloor$. As explained above, equivalence classes $[0], \ldots, [\alpha-1]$ have roughly the same amount of elements of $1, \ldots, n+k-1$. To be precise, for every $i \in \{0, \ldots, \alpha-1\}$, $\#[i] \in \{\beta, \beta+1\}$. (In particular, if $n+k-1$ is multiple of α, then for all $i \in \{0, \ldots, \alpha-1\}$, $\#[i] = \beta$; otherwise there is at least one $i \in \{0, \ldots, \alpha-1\}$ such that $\#[i] = \beta+1$.) Clearly we have $\sum_{j=0}^{\alpha-1} \#[j] = n+k-1$, thus, for any $i \in \{0, \ldots, \alpha-1\}$,

$$\sum_{j=0, j\neq i}^{\alpha-1} \#[j] = \sum_{j=0}^{\alpha-1} \#[j] - \#[i] \leq n+k-1-\beta,$$

because $\#[i] \in \{\beta, \beta+1\}$. Therefore, if we prove that $\beta \geq k$, then inequality (9) holds.

We now prove that $\beta \geq k$. We identify two cases: (a) n is multiple of k, i.e., $n = xk$, for some $x \geq 1$, and (b) n is not multiple of k, i.e., $n = xk + y$, for some $x \geq 0$ and $1 \leq y \leq k-1$. For case (a) we have that $\alpha = x$, and thus $\beta = \lfloor \frac{n+k-1}{\alpha} \rfloor = \lfloor k + \frac{k-1}{x} \rfloor \geq k$. For case (b) we have that $\alpha = x+1$, and hence $\beta = \lfloor \frac{n+k-1}{\alpha} \rfloor = \lfloor \frac{xk+y+k-1}{x+1} \rfloor = \lfloor \frac{k(x+1)+y-1}{x+1} \rfloor = \lfloor k + \frac{y-1}{x+1} \rfloor \geq k$. Inequality (9) follows.

Using inequality (9), the correctness proof of $\mathcal{A}$ is simple. Suppose, by contradiction, that there is an execution in which an element x of $\{1, \ldots, \lceil \frac{n}{k} \rceil\}$ is not decided by at least one process. Therefore, it must be that the n processes of the system got values from the $\langle n, n+k-1, 0, 1\rangle$-GSB object, that do not belong to the equivalence class $[x-1]$. Thus, $\sum_{j=0, j\neq i}^{\alpha-1} \#[j] \geq n$, which contradicts inequality (9). The theorem follows. $\square_{Theorem\ 14}$

5 Conclusion

Significant efforts have been devoted in the past to the elusive question of understanding the relative computability power of subconsensus tasks in a wait-free setting. These efforts have lead to the discovery of new algorithmic ideas and sophisticated algebraic topology techniques for impossibility results. Yet, essentially only one relation was known for specific values of n: $(2n-2)$-renaming is strictly weaker than $(n, n-1)$-SA for odd n. In this paper we have significantly developed the map of computability relations among renaming, set agreement and other GSB tasks. The map with our results and previous results, is in Figure 1.

Among other results, we have showed that even the most powerful non-perfect symmetry breaking problem, namely non-adaptive $(n+1)$-renaming is not powerful enough to solve the easiest agreement problem, namely $(n, n-1)$-SA. Hence the question: can non-adaptive perfect renaming, i.e. n-renaming, be used to solve $(n, n-1)$-SA? If so, which is the value of $k < n-1$ such that (n, k)-SA can be solved from perfect renaming?

Some of the previous impossibility results have been proved in the iterated model IM, hence the question: can we extend the simulation of [16] to generalize the results to the non-iterated model? More precisely, [16] proves that IM with 01-*tasks* [13] is equivalent to the snapshot model with 01-tasks, which is sufficient to extend the result in [17] to a non-iterated model. An open issue is then to extend the result in [16] to include GSB tasks, and hence generalize our results to a non-iterated model.

References

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

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

[3] 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.

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

[5] Borowsky E. and Gafni E., A Simple Algorithmically Reasoned Characterization of Wait-Free Computations (Extended Abstract). *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC'97)*, pp. 189-198, 1997.

[6] Castañeda A., Herlihy M. and Rajsbaum S. An Equivariance Theorem with Applications to Renaming. *Tech Report #1975*, IRISA, Université de Rennes 1 (France), 2011.

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

[8] Castañeda A., Rajsbaum S. and Raynal M., The Renaming Problem in Shared Memory Systems: an Introduction. *Computer Science Review*, to appear, 2011.

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

[10] Chaudhuri S., and Reiners, P., Understanding the Set Consensus Partial Order Using the Borowsky-Gafni Simulation (Extended Abstract). *10th Int'l Workshop on Distributed Algorithms (WDAG'96)*, Springer Verlag LNCS #1171, pp. 362-379, 1996.

[11] 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.

[12] Gafni E., Renaming with k-set Consensus: an Optimal Algorithm in $n+k-1$ Slots. *Proc. 10th Int'l Conference On Principles Of Distributed Systems (OPODIS'06)*, Springer Verlag LNCS #4305, pp. 36-44, 2006.

[13] 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.

[14] Gafni E. and Kuznetsov P., N-Consensus is the Second Strongest Object for N+1 Processes. *Proc. 11th Int'l Conference On Principle Of Distributed Systems (OPODIS'07)*, Springer Verlag LNCS #4878, pp. 260-273, 2007.

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

[16] Gafni E. and Rajsbaum S., Distributed Programming with Tasks. *Proc. 10th In'l Conference on Principles of Distributed Systems (OPODIS'10)*, Springer Verlag LNCS #6490, pp. 205-218, 2010.

[17] 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.

[18] Gafni E., Raynal M. and Travers C., Test&Set, Adaptive Renaming and Set Agreement: a Guided Visit to Asynchronous Computability. *Proc. 26th Int'l IEEE Symposium on Reliable Distributed Systems (SRDS 2007)*, IEEE Press, pp. 93-102, 2007.

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

[20] Herlihy M.P. and Rajsbaum S., Set Consensus Using Arbitrary Objects (Preliminary Version). *Proc. 13th Annual ACM Symposium on Principles on Distributed Computing (PODC'94)*, ACM Press, pp. 324-333, 1994.

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

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

[23] Imbs D., Rajsbaum S. and Raynal M., The Universe of Symmetry Breaking Tasks. *Submitted to publication*, Tech report #1965, IRISA, Université de Rennes 1 (France), 2011.

[24] Mostéfaoui M., Raynal M., and Travers C., Exploring Gafni's Reduction Land: from Ω^k to Wait-free adaptive $(2p - \lceil \frac{p}{k} \rceil)$-renaming via k-set Agreement. *Proc. 20th Int'l Symposium on Distributed Computing (DISC'09)*, Springer Verlag LNCS #4167, pp. 1-15, 2006.

[25] Rajsbaum S. Iterated Shared Memory Models. *Proc. 9th Latin American Symposium on Theoretical Informatics (LATIN'10)*, Springer Verlag LNCS #6034, pp. 407-416, 2010.

[26] 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.