Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 283 (2012) 29–51

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

An Introduction to the Topological Theory of Distributed Computing with Safe-consensus

Rodolfo Conde[1,2] and Sergio Rajsbaum[3,4]

Instituto de Matemáticas
Universidad Nacional Autónoma de México
Ciudad Universitaria, México D.F. 04510, México

Abstract

The theory of distributed computing shares a deep and fascinating connection with combinatorial and algebraic topology. One of the key ideas that facilitates the development of the topological theory of distributed computing is the use of *iterated* shared memory models. In such a model processes communicate through a sequence of shared objects. Processes access the sequence of objects, one-by-one, in the same order and asynchronously. Each process accesses each shared object only once. In the most basic form of an iterated model, any number of processes can crash, and the shared objects are snapshot objects. A process can write a value to such an object, and gets back a snapshot of its contents.
The purpose of this paper is to give an introduction to this research area, using an iterated model based on the *safe-consensus* task (Afek, Gafni and Lieber, DISC'09). In a safe-consensus task, the validity condition of consensus is weakened as follows. If the first process to invoke an object solving a safe-consensus task returns before any other process invokes it, then the process gets back its own input; otherwise the value returned by the task can be arbitrary. As with consensus, the agreement requirement is that always the same value is returned to all processes.
A safe-consensus-based iterated model is described in detail. It is explained how its runs can be described with simplicial complexes. The usefulness of the iterated memory model for the topological theory of distributed computing is exhibited by presenting some new results (with very clean and well structured proofs) about the solvability of the (n, k)-set agreement task. Throughout the paper, the main ideas are explained with figures and intuitive examples.

Keywords: distributed system, wait-free, set agrement, consensus, safe-consensus, topology.

1 Introduction

The theory of distributed computing is an actively developed field of computer science that shares a deep and fascinating connection with combinatorial and algebraic topology. One of the key ideas that facilitates the development of the topological

[1] Supported by a CONACyT grant.

[2] Email: rodolfo@math.unam.mx

[3] Supported by DGAPA-PAPIME and PAPIIT grants.

[4] Email: rajsbaum@math.unam.mx

doi:10.1016/j.entcs.2012.05.004

theory of distributed computing is the use of *iterated* shared memory models, introduced in [10]. In such a model processes communicate through a sequence of shared objects. Processes access the sequence of objects, one-by-one, in the same order and asynchronously. Each process accesses each shared object only once.

In the most basic form of an iterated model, any number of processes can crash, and the shared objects are snapshot objects. A process can write a value to such an object, and gets back a snapshot of its contents. It is known that this model is equivalent to the standard wait-free read/write shared memory model [10,15], but its runs are more structured and easier to analyse than the runs in the standard shared memory model. The recursive nature of the iterated shared memory model was instrumental for the results in [10] and for the proof of the *Asynchronous computability theorem* of [19]. This theorem uncovered the intimate connection that exists between topology and distributed computing. Extensions of the basic iterated model have also been studied, where the processes communicate through a sequence of objects more powerful than snapshot objects [16] or where the asynchrony of the system is limited to model failure detectors [22]. For an overview of the iterated approach see [21].

The purpose of this paper is to give an introduction to this research area (for the non-specialist), using an iterated model based on the *safe-consensus* task of Afek, Gafni and Lieber [2]. In a safe-consensus task, the validity condition of *consensus* [13] is weakened as follows. If the first process to invoke an object solving a safe-consensus task returns before any other process invokes it, then the process gets back its own input; otherwise the value returned by the task can be arbitrary. As with consensus, the agreement requirement is that always the same value is returned to all processes. The safe-consensus task was introduced in [2] (as a generalization of the consensus problem) to show the equivalence of the g-tight-group-renaming task of [3] and the consensus task for g processes. The paper also proves that the safe-consensus task is as powerful as consensus.

We define an iterated safe-consensus-based shared memory model in two ways: the classical fashion and the "topological way". After that, we show its usefulness, and its connection with topology by exhibiting some new results (with easy, well structured proofs) about the solvability of consensus and (n, k)-set agreement [11] (where n is the number of processes). While in the consensus task processes agree on at most one value, the (n, k)-set agreement task allows processes to agree on at most k different values.

The iterated model studied in this paper, is an extension of the iterated model of [10] with the power of *safe-consensus objects.* A *safe-consensus object* is a shared object that receives input values from the processes and returns to the processes output values consistent with the safe-consensus task specification (Section 3). The new model allows the processes to communicate by using a sequence of snapshot objects and safe-consensus objects. As with any iterated model, computation proceeds in iterations, accessing copies of the objects, asynchronously, and in the same order. In each iteration, each process first writes to the shared memory, then it invokes a safe-consensus object and finally it takes a snapshot of the shared memory. As the purpose of this paper is to give an easy to read introduction to the area, we will put an additional restriction in our new extended model: Each safe-consensus object in the sequence must be invoked by all the processes (and in the same order). We call this model of computation the *iterated shared memory with full safe-consensus objects* (IFSC) model.

Specifically, the results we present are:

- The $(n, n-1)$-set agreement task can be implemented in the IFSC model using only one safe-consensus object (with a simple one-round protocol, Theorem 3.2);
- It is impossible to solve the consensus problem for n processes in the IFSC model (Theorem 3.6).

These results say that the IFSC model is indeed more powerful than the basic iterated model, because $(n, n-1)$-set agreement cannot be solved using only shared memory [8,18,23]. But the IFSC model still has limitations as it cannot solve consensus (while consensus is solvable using safe-consensus objects [2] without restrictions). This impossibility does not come from the fact that we are working in an iterated model, rather, it is caused by the requirement that processes access the same safe-consensus object altogether in each iteration.

Also, in connection with the topological theory of distributed computing, to represent executions of protocols in the iterated model (and in the extended IFSC model), we will have something to say about Theorem 3.6. It is known that the protocol complexes in the iterated shared memory model are connected and because of this, it is impossible to solve consensus [7,19]. In this paper, we argue that the protocol complexes in the IFSC model are disconnected, yet, these protocols cannot solve consensus. The discussion about Theorem 3.6 is in Section 3.

In summary, our aims in this introductory paper include explaining:

(i) How to analyze protocol complex connectivity in an iterated model;

(ii) how does the power of the communication objects affects the connectivity of a protocol complex;

(iii) and what are the consequences of connectivity for task solvability.

As mentioned above, the iterated shared memory model has been extended with objects more powerful than read/write registers [16], but these objects were at most as powerful as the set agreement task. To our knowledge, this is the first time in which an attempt is made to study an iterated model by adding objects as powerful as the consensus problem. Nevertheless, we stress that our intention in this paper is not to provide new results, but to give an introduction to the area. In a sequel to this paper [12] we study in more depth iterated models extended with safe-consensus objects. One of these models is the IFSC model.

The outline of the paper is as follows. In Section 2 we give an introduction to the basic concepts of the iterated shared memory model (in the usual combinatorial way), and then we show how to represent the behavior of protocols in terms of combinatorial topology, using simplicial complexes. Here we also give the definition of the (n, k)-set agreement task [11], the task in which we will focus our attention. In Section 3 we extend the basic iterated model of shared memory, adding the power of safe-consensus objects to obtain the IFSC model described above. We study the IFSC model using the tools introduced in Section 2. Throughout the paper, we explain the main ideas behind concepts and proofs with figures and simple examples. Section 4 contains our concluding remarks.

2 The iterated shared memory model

We now introduce the iterated shared memory model of distributed computing [10], and explain how the runs of protocols in this model have a well behaved geometric structure given in terms of simplicial complexes.

2.1 Basic model of computation

Our formal model is standard, see e.g. [6], so we do not state it here in detail. A *system* consists of n processes $p_1, \ldots, p_n$. A *process* is a deterministic state machine, which has a (possible infinite) set of *local states*, including a subset called the *initial states* and a subset called the *output states*. Processes communicate by means of a *shared memory*, structured as a sequence of arrays $SM^{(i)}[1 \cdots n]$ $(i \geqslant 0)$ of a finite number of *single-writer multi-reader atomic registers*. Each register may attain values from some domain, which includes a special "undefined" value $\perp$. We make no assumptions about the size of the registers, and therefore we may assume that each process p_i can write its entire state in a single register. Each shared memory $SM^{(i)}$ provides two atomic operations that can be used at most once by a process.

- $SM^{(i)}.\mathsf{write}()$: writes some value to the register $SM^{(i)}[j]$, where j is the id of process p_j.
- $SM^{(i)}.\mathsf{snapshot}()$: returns a copy of the whole shared memory array $SM^{(i)}$.

We assume w.l.o.g. that a process always alternated write and snapshot operations. Notice that the snapshot operation can be implemented in read/write shared memory [1].

A (system) state consists of the local states of the processes and all the registers in the shared memory. Formally, a *state* is a vector

$$S = \langle s_1, \ldots, s_n; SM \rangle,$$

where s_i is the local state of process p_i and SM is the shared memory of the system. An *initial state* is a state in which every local state is an initial local state and all register in the shared memory are set to $\perp$. A *decision state* is a state in which all local states are output states.

Events and round schedules

An *event* of the system is performed by a single process p_i, which applies only one of the following actions: a write (W) or read (R) operation (i.e. a snapshot). Any of these operations may be preceded/followed by some local computation, formally a change of the process to its next local state. It is convenient to consider events performed concurrently. If E is any event and $p_{i_1}, \ldots, p_{i_k}$ are processes, then we denote the fact that $p_{i_1}, \ldots, p_{i_k}$ execute concurrently the event E by $\mathsf{E}(i_1, \ldots, i_k)$. A *round schedule* is a finite sequence of the form

$$\mathsf{E_1}(j_{11}, \ldots, j_{1p_1}), \ldots, \mathsf{E_r}(j_{r1}, \ldots, j_{rp_r}),$$

that encodes the way in which processes perform the events $\mathsf{E}_1, \ldots, \mathsf{E}_r$. For example the round schedule given by

$$\mathsf{W}(1,3), \mathsf{R}(1,3), \mathsf{W}(2), \mathsf{R}(2),$$

means that processes p_1, p_3 perform the write and read events concurrently; after that, p_2 executes solo its read/write events. Similarly, the round schedule $\mathsf{W}(1,2,3), \mathsf{R}(1,2,3)$ says that p_1, p_2, p_3 execute concurrently the write/read operations. Let $\bar{n} = \{1, \ldots, n\}$. Then we denote $\mathsf{E}(1, \ldots, n)$ by $\mathsf{E}(\overline{n})$; if $A \subset \overline{n}$ and $\overline{n} - A = \{i_1, \ldots, i_q\}$, $\mathsf{E}(\overline{n} - A)$ denotes $\mathsf{E}(i_1, \ldots, i_q)$; when $A = \{i\}$, $\mathsf{E}(\overline{n} - A)$ is $\mathsf{E}(\overline{n} - i)$.

Decision tasks

A decision task Δ is a relation that has a domain $\mathcal{I}$ of input values and a domain $\mathcal{O}$ of output values; Δ specifies for each assignment of the inputs to processes on which outputs processes can decide. Examples of tasks includes *consensus* [13], *renaming* [9,3] and the *set agreement* task [11] (to be defined later).

```
(1) init $r_i \leftarrow 0$; $sm_i \leftarrow input_i$; $dec_i \leftarrow \perp$;

(2) loop forever
(3)       $r_i \leftarrow r_i + 1$;
(4)       $SM^{(r_i)}$.write($sm_i$);
(5)       $sm_i \leftarrow SM^{(r_i)}$.snapshot();

(6)       if ($dec_i = \perp$) then
(7)             $dec_i \leftarrow \delta(sm_i)$;
(8)       end if
(9) end loop
```

Figure 1. General form of a protocol in the iterated model (code for p_i)

2.2 The basic iterated model

The state machine of each process p_i models a *local protocol* $\mathcal{A}_i$, that determines the steps taken by p_i. We assume that all local protocols are identical; i.e. Processes have the same state machine. A *protocol* is a collection $\mathcal{A}$ of local protocols $\mathcal{A}_1, \ldots, \mathcal{A}_n$. In this paper, we are interested in protocols in the iterated read/write shared memory model of distributed computing. From now on we just call it the *basic iterated model.* For the sake of simplicity, we give protocols specifications using pseudocode and we establish the following conventions: A lowercase variable denotes a local variable, with a subindex that indicates to which process it belongs; the shared memory (which is visible to all processes) is denoted with uppercase letters. Figure 1 shows the general form of a protocol in the iterated model.

An *execution* of a protocol $\mathcal{A}$ is a finite or infinite alternating sequence of states and round schedules $S_0, \pi_1, \ldots, S_k, \pi_{k+1}, \ldots$, where S_0 is an initial state and S_k is the resulting state of applying the sequence of events performed by the processes in the way described by π_k. An *r-round partial execution* of $\mathcal{A}$ is a finite execution of $\mathcal{A}$ of the form $S_0, \pi_1, \ldots, S_{r-1}, \pi_r, S_r$, that is, an execution of $\mathcal{A}$ until the end of round r. A state P is said to be *reachable in* $\mathcal{A}$ if there exists an r-round partial execution of $\mathcal{A}$ ($r \geqslant 0$) that ends in the state P and when there is no confusion, we just say that S is reachable.

We identify two special components of each process' states: an input and an output. It is assumed that initial states differ only in the value of the input component; moreover, the input component never changes. The protocol cannot overwrite the output, it is initially $\perp$; once a non-$\perp$ value is written to the output component of the state, it never changes; when this occurs, we say that the process *decides.* The output states are those with non-$\perp$ output value.

A protocol *solves* a decision task Δ if any finite execution α can be extended to an execution α' in which all processes decide on values which are allowable (accordingly to Δ) for the inputs in α. Because the outputs cannot be overwritten, if a process has decided on a value in α, it must have the same output in α'. This means that outputs already written by the processes can be completed to outputs for all processes that are permissible for the inputs in α.

A protocol is *wait-free* if in any execution of it, a process either it has a finite number of events or it decides. This implies that if a process has an infinite number of events, it must decide after a finite number of events.

We do not require the processes to halt; they solve the decision task and decide by writing to the output component; processes can continue to participate. We typically consider the behavior of a process until it decides, and therefore, the above distinction does not matter.

Properties of states

Let $\mathcal{A}$ be a protocol in the iterated model and let S, R be states, we say that R is a *successor* of S in $\mathcal{A}$, if there exists an execution α of $\mathcal{A}$ such that

$$\alpha = S_0, \pi_1, \ldots, S_r = S, \pi_{r+1}, \ldots, \pi_{r+k}, S_{r+k} = R, \ldots,$$

i.e., starting from S, we can run the protocol $\mathcal{A}$ k rounds (for some $k \geqslant 0$) such that the system enters state R. If π is any round schedule and S is a state, the successor of S in $\mathcal{A}$ obtained by running the protocol (starting in the state S) one iteration with the round schedule π is denoted by $S \cdot \pi$.

Two states S, P are said to be *adjacent* if there exists a non-empty subset $X \subseteq \overline{n}$ such that all processes with ids in X have the same local state in both S and P. That is, for each $i \in X$, p_i cannot *distinguish* between S and P. We denote this by $S \overset{X}{\sim} P$ and when $X = \{i\}$ we use the notation $S \overset{i}{\sim} P$. States S and P are *connected* if we can find a sequence of states $S = P_1, \ldots, P_r = P$ such that for all j with $1 \leqslant j \leqslant r-1$, P_j and P_{j+1} are adjacent.

Equivalence of the iterated model with the standard model

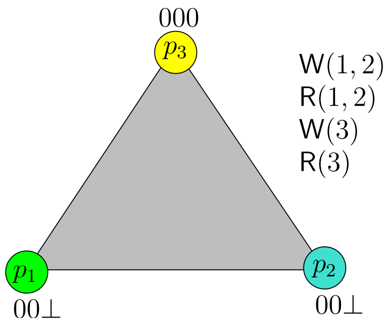
It would seem that the iterated model that we have defined is too restrictive: a process cannot go back and read again the same shared array. Moreover, all processes must access all the shared arrays in the same order. Because of this, one may think it does not have full generality when compared with the more standard, non-iterated wait-free shared memory model, that does not have the restrictions imposed to protocols in the iterated model. The crucial question is: Does there exists a task that is solvable in the wait-free standard model and it is not solvable in the iterated model? The answer is no. Every task that is solvable in the wait-free standard model is also solvable in the iterated model. This is proved using an algorithm that simulates the standard model in the iterated model, first in [10], and more recently in [15]. Therefore, there is no loss of generality (for computability purposes) in considering only protocols in the iterated model.

2.3 The geometry of the Iterated model

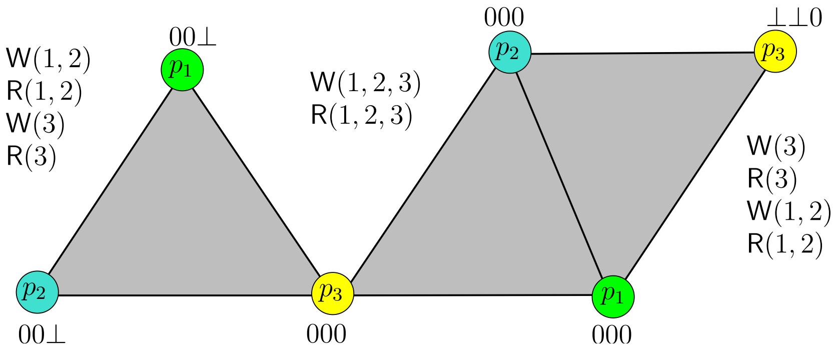
The framework we have described to study distributed algorithms can be given in geometric terms, using simplicial complexes. We assume that the reader is familiar with basic concepts from combinatorial topology [5,4].

We begin with an example, let $n = 3$ and suppose processes p_1, p_2 and p_3 execute a protocol $\mathcal{A}$ in the iterated model with their input values all equal to 0. In the first round of $\mathcal{A}$, there are several possible ways for the processes to execute their actions of reading and writing to the shared memory; one possibility is that p_1 and p_2 execute concurrently the basic operations and later, p_3 executes the same steps. This is represented by the round schedule $\mathsf{W}(1,2), \mathsf{R}(1,2), \mathsf{W}(3), \mathsf{R}(3)$. At the end of the round, the local view of the shared memory of p_1 and p_2 only contains the values these processes wrote; they cannot see the value written by process p_3 because they executed faster that p_3, which wrote its information in the shared memory after p_1 and p_2 executed their reads (snapshots) operations[5]. But p_3's local view of the memory contains the data of the three processes, so that p_3 can see all the information written by p_1 and p_2. This (system) state (which is reachable in $\mathcal{A}$) can be represented by a 2-simplex Δ_2 as shown in the following figure.

[5] This models the possibility that p_3 is just running slower that p_1 and p_2 and also it models the situation in which p_3 crashes and does not take any steps at all.



Each vertex of Δ_2 is labeled with the process id and the local state of that process (this includes the local view of the memory). Now consider the case where the three processes execute the shared operations concurrently (this is described with the round schedule $\mathsf{W}(1,2,3), \mathsf{R}(1,2,3)$) and the views they have of the memory at the end of the first round. As in the previous case, we represent this state as a 2-simplex Δ'_2.



This time, all the processes can see each other's input values and because p_1 and p_2 have a different local view of the memory, they can distinguish between the state given by Δ'_2 and the previous one (represented by Δ_2); but p_3 cannot tell the difference, because it has exactly the same information in both states. This is the reason why the simplexes Δ_2 and Δ'_2 have the vertex with p_3's id as a common face, p_3 cannot distinguish between the two states, because in both of them it has the same local state. In a similar way, the simplex that represents the reachable state in $\mathcal{A}$ obtained by executing the first round of the protocol in the way described by $\mathsf{W}(3), \mathsf{R}(3), \mathsf{W}(1,2), \mathsf{R}(1,2)$, is adjacent to Δ'_2 (this time, p_1 and p_2 cannot distinguish).

We can represent all the possible reachable states in the first round of $\mathcal{A}$ as 2-simplexes and construct a simplicial complex, which we call the *1-round protocol complex* of $\mathcal{A}$ (Figure 2 (a)). It is just a subdivided triangle. Each simplex in this complex represents a state S that the system can reach after the processes execute the first round of $\mathcal{A}$ with a round schedule that makes the protocol enter into state S.

In the second round, processes start to execute the protocol with the state they had at the end of the first round, and then all the possible round schedules of the first round can be repeated. If we represent the state that the processes have at

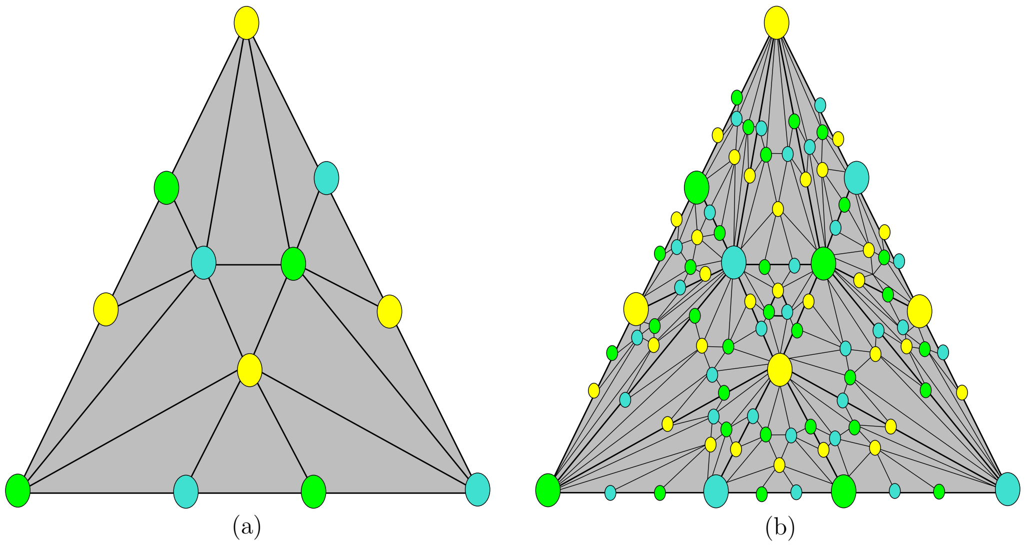


Figure 2. The protocol complex for a 3-process protocol in the iterated model after the execution of the first round (a) and the second round (b).

the beginning of round two as the corresponding 2-simplex of the 1-round protocol complex of $\mathcal{A}$ (that is, as a reachable state of the first round of the protocol) then using the fact that we are working with an iterated model of computing, we can encode all the possible states of the system at the end of round two as a subdivision of the triangle that represents the state that the processes had at the end of round one. This subdivision is identical to the protocol complex of round one, so that we can describe all the possible reachable states in round two of $\mathcal{A}$ as the simplicial complex of Figure 2 (b), this is the *2-round protocol complex of* $\mathcal{A}$. Notice that this complex is also a subdivided triangle, with several subcomplexes isomorphic to the protocol complex of round one. This behavior is going to repeat itself for all subsequent rounds– the *k-round protocol complex* for a protocol in the iterated model for three processes will be a subdivision of the 2-simplex. To obtain the complex of a subsequent round, we just take the complex of the previous round and replace each simplex in it with the complex of Figure 2 (a) and we can see that the k-round protocol complex has a simple and elegant recursive structure.

In general, if we have $n+1$ processes, each possible initial or final local state of a process is modeled as a vertex $v = (i, sm_i)$, a pair consisting of a process id i and the local state sm_i of process p_i. We say that the vertex is *coloured* with the process id. A set of $n+1$ mutually compatible initial or final local states is represented as an n-simplex Δ_n and with this we model a possible system state. Given a protocol $\mathcal{A}$ in the iterated model, where each process p_i receives the input value v_i $(i = 1, \dots, n+1)$, for each round number $r \geqslant 0$ all the possible reachable states in $\mathcal{A}$ at the end of round r are represented as a n dimensional complex, the *r-round protocol complex* (for brevity, we just say the "protocol complex"), $\mathcal{P}_{\mathcal{A}}$, in which each vertex is labeled with a process id and that process state at the end of round r. Thus, each simplex in $\mathcal{P}_{\mathcal{A}}$ corresponds to an equivalence class of executions of $\mathcal{A}$ that "look the same" to the processes at its vertices. It is argued in [10] that $\mathcal{P}_{\mathcal{A}}$ is always a subdivided n-simplex.

There is more information about the protocol complex in [19]. In fact, the definition we have presented of the r-round protocol complex $\mathcal{P}_{\mathcal{A}}$ (associated with the input values that the processes have when they begin executing the protocol $\mathcal{A}$) is closely related with the notion of a *span* in [19, Definition 4.4, page 884]. Herlihy and Shavit define the protocol complex as a bigger geometric structure which depends on all the possible input values of processes and all possible executions of a protocol $\mathcal{A}$. However, for most cases, it is sufficient to work with the definition of protocol complex we have given. We find our definition sufficient for the purposes of this introductory paper.

2.4 Set agreement tasks

We are interested in the (n, k)-set agreement task, where n is the number of processes and $k < n$. This task was proposed the first time in [11] and it has been fundamental in the study of distributed computing. It is described as follows:

The (n, k)-set agreement problem. In this task, every process starts with some initial input value taken from a set I ($|I| \geqslant n$) and must output a value such that:

- Termination: Each process must eventually output some value.
- k-Agreement: The set of output values from all processes must be of size at most k.
- Validity: If some process outputs v, then v is the initial input of some process.

The set agreement task is a natural generalization of *consensus*, which corresponds to the $(n, 1)$-set agreement task. That is, processes must agree on a unique output value. Before the set agreement task was defined, it was well known that the consensus problem is not solvable in the presence of even only one faulty process [13]. This impossibility result uses simple graph connectivity arguments. But since the (n, k)-set agreement task was conceived, it was an open question whether it could be solved in the wait-free shared memory model (with the parameters $2 \leqslant k < n$), until 1993 when three independent teams [8,23,18] showed that there is no wait-free protocol that can solve set agreement.

Theorem 2.1 ([8,23,18]) *For $1 \leqslant k < n$, the (n, k)-set agreement task has no wait-free read/write solution in the iterated shared memory model.*

As the iterated model is equivalent to the usual standard wait-free shared memory model, Theorem 2.1 implies the impossibility to solve the set agreement task in the standard model. It is worth noticing that the set agreement task is not the only distributed problem in which topological techniques have been useful. Other examples are musical benches [14], renaming [19] and loop agreement [17] among others.

3 An enrichment of the iterated model

In this section, we add more powerful shared objects to the iterated model, and study the solvability of the set agreement task in this extended model.

3.1 The extended iterated model with safe-consensus

The objects we will add to the iterated model are based on a variant of the consensus problem, which is called *safe-consensus.*

The safe-consensus task. In this task, every process starts with some initial input value taken from a set I and must output a value such that:

- Termination: Each process must eventually output some value.
- Agreement: All processes output the same value.
- Validity: (1) If a process p_i starts executing the task and outputs before any other process starts executing the task, then the task's output is p_i's proposed input value. (2) Otherwise, if two or more processes initially access the safe-consensus task concurrently, then the task can return any value from a countable set V such that I is a proper subset of V.

The safe-consensus task is the result of weakening the validity condition of consensus. It was first proposed by Yehuda, Gafni and Lieber [2]; they use it to show that the g-tight-group-renaming task [3] is as powerful as consensus for g processes. We can now define new objects to extend the basic iterated model.

A *safe-consensus object* is a shared object that can be used by any number of processes. The object receives an input value from each process that invokes it, and returns to all the processes an output value that satisfies the Agreement and Validity condition of the safe-consensus task. In other words, a safe-consensus object is like a "black box" that the processes can use to solve instances of the safe-consensus task. The method of using distributed tasks as black boxes inside protocols is a standard way to study the relative computational power of distributed tasks (i.e. if one task is weaker than another).

It can be seen that the safe-consensus shared objects are primitives more powerful than the read/write shared memory. This is because in [2] the authors give two protocols that solve the consensus problem for n processes in a model of distributed computing that extends the standard shared memory model with safe-consensus objects (without any restriction in the way that the processes use the safe-consensus objects) and this says that the safe-consensus task is as powerful as consensus. Remember that in this paper, in order to make a clear exposition of the field, we use the safe-consensus objects by imposing some rules in the way in which processes invoke the shared objects.

More precisely, we add to the basic iterated model defined in Section 2.2, an infinite array of safe-consensus objects; each of these provides an *exec* method that takes one parameter as input value and returns to all participating processes a unique value, satisfying the properties of the safe-consensus task. We assume that the input value that processes feed to the safe-consensus objects is their own ids[6]. For the purposes of this paper, we also assume that in each iteration, all the processes invoke the same safe-consensus object, that is, there is only one safe-consensus object in each round and it is used by all processes. We call this model of computation the *Iterated shared memory with full safe-consensus objects* (IFSC) model (Figure 3). Finally, we extend the set of events of the iterated model with the event of a call to a safe-consensus object. This event will be denoted by S.

```
(1)  init $r_i \leftarrow 0$; $sm_i \leftarrow input_i$; $dec_i \leftarrow \perp$; $scret_i \leftarrow \perp$;

(2)  loop forever
(3)      $r_i \leftarrow r_i + 1$;
(4)      $SM^{(r_i)}$.write($sm_i, scret_i$);
(5)      $scret_i \leftarrow safe\text{-}consensus.exec(id)$;
(6)      $sm_i \leftarrow SM^{(r_i)}$.snapshot();

(7)      if ($dec_i = \perp$) then
(8)          $dec_i \leftarrow \delta(sm_i, scret_i)$;
(9)      end if
(10) end loop
```

Figure 3. General form of a protocol in the IFSC model (code for p_i)

Before continuing with our exposition, we would like to make some remarks concerning the model of distributed computing that we have defined. Notice in Figure 3 that the calls to the safe-consensus objects have been placed between the write and read (snapshot) shared memory operations. In the sequel to this paper [12], we investigate different iterated models which arise from extending the basic iterated model with safe-consensus objects. One issue we consider is precisely where to place the calls to the safe-consensus objects. As we explain in [12], depending on where we put the calls to safe-consensus objects, we can obtain quite different models. In fact, one of the models investigated in [12] is an extension of the IFSC model.

3.2 The protocol complex with safe-consensus

After we add to the basic iterated model the power to use safe-consensus objects, one of the first questions that we would like to answer is: What happens to the protocol complex in the extended model ? Are the topological properties unchanged ? In order to answer this question, the first thing we must do is to formally define the protocol complex for protocols in the new IFSC model. If $\mathcal{A}$ is a protocol in the IFSC model for n processes, the protocol complex $\mathcal{S}_{\mathcal{A}}$ of $\mathcal{A}$ is defined in the same way as we defined protocol complexes in Section 2.3, the only thing that changes is the view associated to a vertex. A vertex of $\mathcal{S}_{\mathcal{A}}$ is a tuple $v = (i, sm_i, val)$, where i is the id of process p_i, sm_i is the local state of p_i and val is the return value of the safe-consensus object invoked by all processes.

In Figure 4 we have a drawing of a part of the protocol complex $\mathcal{S}_{\mathcal{A}}$ of a one-round execution of a protocol $\mathcal{A}$ for three processes in the IFSC model. The complete

[6] It is not hard to see that this assumption is done without loss of generality.

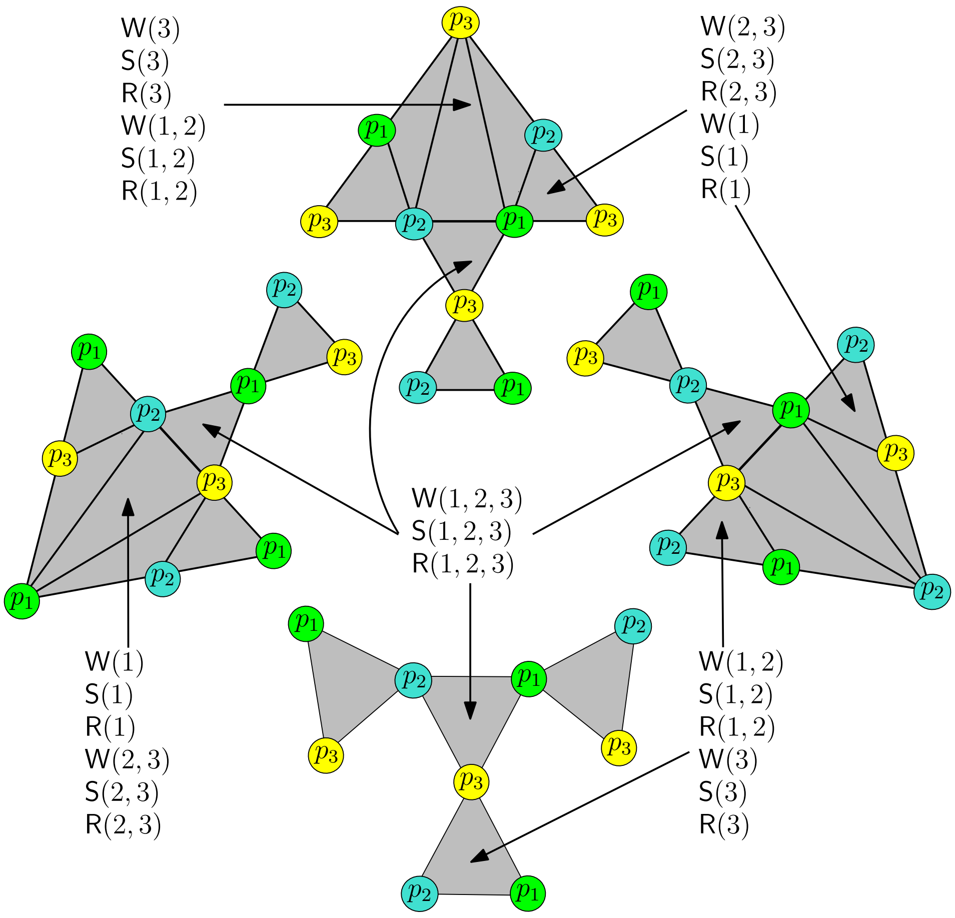


Figure 4. Part of the one round Protocol complex of a protocol in the IFSC model. The entire complex contains a countable number of copies of the subcomplex at the bottom. 3.1).

complex is a disconnected space with an infinite number of connected components. Clearly, this complex is not the same complex of Figure 2. Each connected component is associated with an output value of the safe-consensus task. To see why this is true, let us take a closer look at the subcomplex, say $\mathcal{T}_3$, on top of Figure 4, it has a 2-simplex that represents a state in the first round of $\mathcal{A}$ that is reachable by means of the round schedule $\mathsf{W}(3), \mathsf{S}(3), \mathsf{R}(3), \mathsf{W}(1,2), \mathsf{S}(1,2), \mathsf{R}(1,2)$. In this execution, process p_3 is faster that p_1 and p_2, thus it executed the safe-consensus object in solo, hence by the Validity condition of the safe-consensus task, the shared object returns the valid input value of 3 to all processes. Now, if we consider the 2-simplex of $\mathcal{T}_3$ representing the state reachable through the schedule $\mathsf{W}(2,3), \mathsf{S}(2,3), \mathsf{R}(2,3), \mathsf{W}(1), \mathsf{S}(1), \mathsf{R}(1)$ then we know that p_2 and p_3 were faster that p_1 and executed the safe-consensus object concurrently and by the Validity condition of the safe-consensus task, the shared object can return any value to all processes, so that there exists the possibility that the return value of the safe-consensus object is 3. With a similar analysis of the other simplexes in $\mathcal{T}_3$, we can conclude that the safe-consensus value given in the vertices of every 2-simplex of $\mathcal{T}_3$

is precisely 3. This subcomplex cannot have simplexes with a safe-consensus value other that 3. For example, it is easy to show that the simplex representing the state obtained with the execution of $\mathcal{A}$ given by $\mathsf{W}(1), \mathsf{S}(1), \mathsf{R}(1), \mathsf{W}(2,3), \mathsf{S}(2,3), \mathsf{R}(2,3)$ cannot be adjacent to any simplex of $\mathcal{T}_3$ (because of the different safe-consensus values). If we now take the subcomplex $\mathcal{T}_1$ in the leftmost part of the figure and the subcomplex $\mathcal{T}_2$ in the rightmost part (both of them are isomorphic to $\mathcal{T}_3$), we can prove that $\mathcal{T}_1$ contains only simplexes with vertices of safe-consensus value equal to 1 and all the simplexes of $\mathcal{T}_2$ have vertices with 2 as the safe-consensus value.

What about the subcomplex B_x at the bottom of the figure ? It contains simplexes that represent states in which the safe-consensus object returns an invalid output value $x \neq 1, 2, 3$. All the states represented in B_x come from executions in which at least two processes invoke the safe-consensus task concurrently and the value the processes obtain from the shared object is precisely x. But there are a countable number of invalid output values that the task can return to the processes, so that there should be in the complex of Figure 4 a subcomplex B_β (isomorphic to B_x) for each possible invalid value β. For simplicity, we only show one of these subcomplexes. In summary, the protocol complex $\mathcal{S}_\mathcal{A}$ of the first round of the protocol is disconnected, with an infinite number of connected subcomplexes, one for each possible output value of the safe-consensus task.

We have described the one-round protocol complex of a protocol in the IFSC model. Because we work in an iterated model, in the second (third, fourth and so on) round, the protocol complex is composed of many subcomplexes like $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ and B_x, each simplex of the previous round transforms into a subcomplex like $\mathcal{S}_\mathcal{A}$.

It is not hard to prove that the behavior we have described for 3-processes protocols in the IFSC model, generalizes to protocols with any number of processes $n \neq 3$. The protocol complex of any protocol in the IFSC model is a disconnected complex with an infinite number of connected subcomplexes. In general, the structure of the protocol complex in the IFSC model is not as uniform as the well behaved structure of the protocol complex in the basic iterated model.

3.3 Solving $(n, n-1)$-set agreement tasks with safe-consensus

We now show that the difference between protocols in the basic iterated model and protocols in the extended model with safe-consensus is not only about geometric shapes. In this section, we prove that in the extended model we can solve the $(n, n-1)$-set agreement task for $n \geqslant 2$ processes, a distributed task that we know is unsolvable in the iterated model [18,23,8]. But we also prove that we cannot solve every set agreement task, because n-consensus ($(n, 1)$-set agreement) is impossible to solve in the IFSC model (when $n \geqslant 3$).

A protocol for $(n, n-1)$-set agreement

The protocol in Figure 5 solves $(n, n-1)$-set agreement for $n \geqslant 2$ processes in one round with only one safe-consensus object. We proceed to prove its correctness.

Lemma 3.1 *For any process p_i that executes line 9 of the protocol in Figure 5,*

```
(1)  init $dec_i, scret_i, sm_i \leftarrow \perp$;

(2)  begin
(3)      $SM$.write($input_i$);
(4)      $scret_i \leftarrow safe\text{-}consensus.exec(id)$;
(5)      $sm_i \leftarrow SM$.snapshot();
(6)      if $scret_i \in \{1, \ldots, n\} \wedge sm_i[scret_i] \neq \perp$ then
(7)          $dec_i \leftarrow sm_i[scret_i]$;
(8)      else
(9)          $dec_i \leftarrow sm_i[\alpha_i]$ with $\alpha_i = \min\{\alpha \mid sm_i[\alpha] \neq \perp\}$;
(10)     end if
(11)     decide $dec_i$;
(12) end
```

Figure 5. A $(n, n-1)$-set agreement protocol in the IFSC model (code for p_i)

(a) The cardinality of the set $A_i = \{\alpha \mid sm_i[\alpha] \neq \perp\}$ *is at least 2 and* $\min A_i \neq n$.

(b) If $n \notin A_i$ *then* $\min A_i \neq n - 1$.

Proof (a) Suppose that process p_i executes line 9 of the **if/else** block, then it must be true that $scret_i \notin \{1, \ldots, n\} \vee sm_i[scret_i] = \perp$, so either the safe-consensus object returned an invalid process id or process p_{scret_i} did not write its proposed value to the shared memory before p_i executed the snapshot operation (p_{scret_i} could be slow or perhaps it crashed). In any case, by (2) of the Validity condition of the safe-consensus task, there must exists two processes p_r, p_s that called the safe-consensus object concurrently. These processes wrote their input values to the shared memory before accessing the safe-consensus object, so that when p_i takes a snapshot of the memory, the local array sm_i contains at least two non-$\perp$ values and with this we have that $|A_i| \geqslant 2$. Now suppose that $\alpha_i = \min A_i$ is such that $\alpha_i = n$. There must exists $j \in A_i$ with $j \neq n$, but then $j < n = \alpha_i$ which is a contradiction. Hence, $\alpha_i \neq n$. (b) Given that $n \notin A_i$, we can use a similar argument to that used in (a) and obtain this case. □

Theorem 3.2 *Let* $n \geqslant 2$. *The* $(n, n-1)$*-set agreement problem is solvable in the IFSC model in one round using one safe-consensus object.*

Proof We prove that the protocol in figure 5 solves $(n, n-1)$-set agreement. Trivially, the protocol satisfies the Termination condition of the $(n, n-1)$-set agreement task. Let p_i be any process; after p_i writes to the shared memory its input value, it invokes the unique safe-consensus object and takes a snapshot of the memory, p_i executes the **if/else** block at lines 6-10. First suppose that $scret_i \in \{1, \ldots, n\}$ and $sm_i[scret_i] \neq \perp$. Then process p_{scret_i} wrote to the shared memory its input value before p_i took the snapshot and this implies that $sm_i[scret_i]$ contains a valid proposed input value and this is the decided value of process p_i. On the other hand, if $sm_i[scret_i] = \perp$ or $scret_i \notin \{1, \ldots, n\}$, p_i goes on to execute line 9. By (a) of Lemma 3.1, the set $A_i = \{\alpha \mid sm_i[\alpha] \neq \perp\}$ is not empty, so that there exists a minimum element $\alpha_i \in A_i$ and then when p_i assigns to dec_i the contents of the local register $sm_i[\alpha_i]$, it has a valid proposed input value, so that the decided value of process p_i is correct. Hence, the Validity condition of the $(n, n-1)$-set agreement problem is fulfilled by the protocol.

We now show that the set of values decided by the processes in any execution of the protocol has size no bigger that $n-1$. Suppose that processes $p_{i_1}, \dots, p_{i_r}$ ($r \leqslant n$, some processes may crash) finish an execution of the protocol and let D be the set of values decided by the processes. We argue by cases.

Case 1. Two or more processes executed line 7 of the protocol. Then $|D| \leqslant n-1$, because all processes invoked the same safe-consensus object and by the Validity property of the safe-consensus task, the return value of the shared object is the same for all processes, so that if at least two processes executed line 7, they decided the same value.

Case 2. Only one process p_l executed line 7 of the protocol. Let v be the value that the safe-consensus object returned to all participating processes. Then for process p_l we have that $\mathit{scret}_l = v$ and $\mathit{sm}_l\,[v]$ (which is non-$\perp$) is the value decided by p_l. If $v \in \{1, \dots, n-1\}$ then $|D| \leqslant n-1$, because for every process p_i with $i \neq l$, p_i executed the second part of the **if/else** block and by (a) of Lemma 3.1, p_i could not decide the value proposed by process p_n. On the other hand, if $v = n$, then for all $i \neq l$, $\mathit{scret}_i = n$ and $\mathit{sm}_i\,[\mathit{scret}_i] = \perp$ and these two facts imply that $A_i \subseteq \{1, \dots, n-1\}$. With (b) of Lemma 3.1 we have that $\min A_i \neq n-1$, so that all other processes with ids not equal to l decided at most $n-2$ values, which together with the value decided by p_l, make up for at most $n-1$ different values, therefore $|D| \leqslant n-1$.

Case 3. No process executed line 7 of the protocol. By (a) of Lemma 3.1, for every process p_i we have that $\alpha_i \in \{1, \dots, n-1\}$, so all processes decided at most $n-1$ values.

In any case, we conclude that the set of decided values has size at most $n-1$, thus the protocol satisfies the $(n-1)$-Agreement condition of the $(n, n-1)$-set agreement problem, hence the protocol is correct. □

3.4 Impossibility of consensus in the IFSC model

Theorem 3.2 tell us that the IFSC model is more powerful that the basic iterated model, because in the new model we can solve the $(n, n-1)$-set agreement task. But now we will show that in the IFSC model, we cannot solve consensus, i.e., $(n, 1)$-set agreement.

Trying to solve consensus in distributed systems

The impossibility results for consensus in various models of distributed computing [13,20,7] and the impossibility results for set agreement in the iterated model [19,8,24], have shown that solving consensus in distributed environments is related to connectivity of graphs (0-connectivity of simplicial complexes). Roughly speaking, consensus is not solvable in a given model if for each valid input of the consensus problem, the protocol complex (associated with that input [7]) is connected.

[7] Remember our discussion at the end of Section 2.3 about our definition of protocol complexes and its relation with the definition of a span of [19].

In Theorem 3.6 we will prove that it is not possible to solve the consensus problem for $n \geqslant 3$ processes in the IFSC model, despite the fact that the protocols in this model have disconnected protocol complexes (see Figure 4). We remark that this impossibility comes from the restriction in the use of the safe-consensus objects (all processes invoke the same object in each round) of the IFSC model, and not from the fact that we are working in an iterated model[8].

If $\mathcal{A}$ is a protocol for the consensus problem in the IFSC model and its protocol complex $\mathcal{S}_{\mathcal{A}}$ is disconnected: Why is it impossible for $\mathcal{A}$ to solve consensus? We can always find an execution of $\mathcal{A}$ in which when all the processes have decided their output values, there are at least two processes that decided two distinct values, contradicting the Agreement property of the consensus task. Now, this "bad execution" exists because we can choose $i \in \overline{n}$ and two simplexes $\Delta_i, \Delta_{\overline{n}-i}$ in the complex $\mathcal{S}_{\mathcal{A}}$ such that

- Δ_i represents an execution in which process p_i can see only his own input value (that is, an execution in solo of p_i).
- $\Delta_{\overline{n}-i}$ represents an execution in which the other processes never see p_i's input value.
- The simplexes Δ_i and $\Delta_{\overline{n}-i}$ lie inside a connected component of $\mathcal{S}_{\mathcal{A}}$.

As a consequence of the last point, there is a (graph) path from any vertex of Δ_i to any vertex in $\Delta_{\overline{n}-i}$. Using one of these paths and an argument involving non-distinguishable states, we can prove that the bad execution exists, so that $\mathcal{A}$ cannot solve consensus. Thus we see that this impossibility is due to some kind of "local connectivity" (between a specific pair of vertices) of $\mathcal{S}_{\mathcal{A}}$. Before the formal impossibility proof, we will workout an example with three processes.

Some definitions for consensus protocols

To be able to work with protocols that solve consensus, we need some additional definitions. Let $\mathcal{A}$ be a consensus protocol; if v is a valid input value for processes and S is a reachable state in $\mathcal{A}$, we say that S is *v-valent* if in every execution of $\mathcal{A}$ starting from S, there exists a process that outputs v. S is *univalent* if in every execution starting from S processes always output the same value. If S is not univalent, then S is *bivalent*. Fix $i \in \overline{n}$ and define the round schedules π_i, π_* and $\pi_{\overline{n}-i}$ as

$$(\pi_i) \quad \mathsf{W}(i), \mathsf{S}(i), \mathsf{R}(i), \mathsf{W}(\overline{n}-i), \mathsf{S}(\overline{n}-i), \mathsf{R}(\overline{n}-i),$$

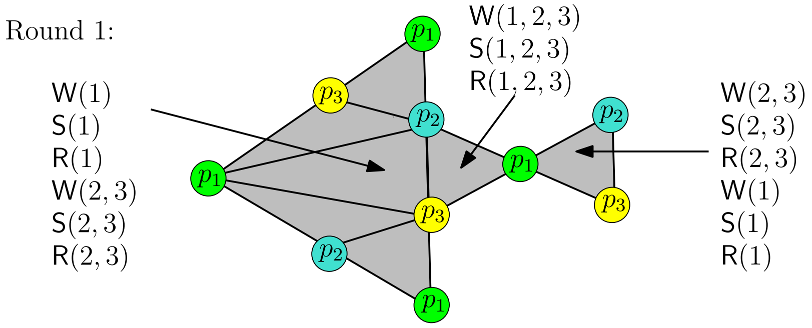
$$(\pi_*) \quad \mathsf{W}(\overline{n}), \mathsf{S}(\overline{n}), \mathsf{R}(\overline{n}),$$

$$(\pi_{\overline{n}-i}) \; \mathsf{W}(\overline{n}-i), \mathsf{S}(\overline{n}-i), \mathsf{R}(\overline{n}-i), \mathsf{W}(i), \mathsf{S}(i), \mathsf{R}(i).$$

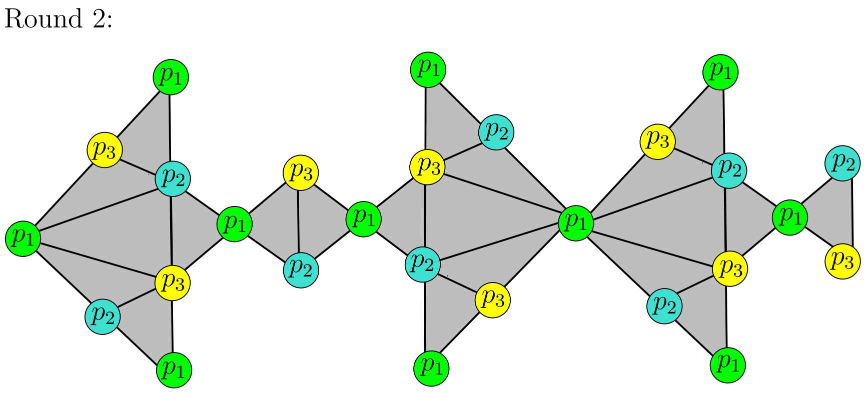
[8] In the sequel to this paper [12] we propose another iterated model with safe-consensus objects (which can be seen as a generalization of the IFSC model) and prove that consensus can be implemented in that model.

An example with three processes

If we take the case of $n = 2$ processes, then by Theorem 3.2, consensus of 2 processes ($(2, 1)$-set agreement problem) is solvable with safe-consensus objects. As we have seen in this section, the protocol complexes of protocols in the IFSC model are disconnected, so that one would expect to be able to solve consensus in this model, but our next results argue otherwise.



$$I \cdot \pi_1 \overset{\{2,3\}}{\sim} I \cdot \pi_* \overset{1}{\sim} I \cdot \pi_{\overline{3}-1}$$



$$(I \cdot \pi_1) \cdot \pi_1 \overset{\{2,3\}}{\sim} (I \cdot \pi_1) \cdot \pi_* \sim \cdots \sim (I \cdot \pi_{\overline{3}-1}) \cdot \pi_* \overset{1}{\sim} (I \cdot \pi_{\overline{3}-1}) \cdot \pi_{\overline{3}-1}$$

Figure 6. Subcomplexes of the first and second round protocol complexes of a protocol in the IFSC model.

Let $\mathcal{A}$ be a protocol for three processes in the IFSC model and assume that $\mathcal{A}$ can solve the consensus problem and for simplicity, suppose that the processes receive as input values their own ids. Let I be the initial state in which the processes begin to execute the protocol. If the processes run $\mathcal{A}$ only one round and then decide a (unique) output value, then we know that the protocol complex $\mathcal{S}_{\mathcal{A}}$ is the space shown in Figure 4. One of the subcomplexes of $\mathcal{S}_{\mathcal{A}}$ is depicted at the top of Figure 6, it represents all the reachable states in the first round of $\mathcal{A}$ for which the output value of the safe-consensus object is 1. The round schedules π_1, π_* and $\pi_{\overline{3}-1}$ give three possible ways to execute the first round of $\mathcal{A}$ and they take the protocol into the states $I \cdot \pi_1, I \cdot \pi_*$ and $I \cdot \pi_{\overline{3}-1}$ respectively and these states are represented in the complex of Figure 6. Because the value returned by the safe-consensus object to all processes in the three states is 1, it is easy to see that p_2 and p_3 cannot distinguish between $I \cdot \pi_1$ and $I \cdot \pi_*$ and p_1 cannot distinguish between $I \cdot \pi_*$ and $I \cdot \pi_{\overline{3}-1}$, thus we have the sequence of connected states

$$I \cdot \pi_1 \overset{\{2,3\}}{\sim} I \cdot \pi_* \overset{1}{\sim} I \cdot \pi_{\overline{3}-1}.$$

If the processes solve consensus in just one round, then these states are decision states. As in the state $I \cdot \pi_1$ process p_1 can see only itself, it has to decide its own input value, which is 1 and by the Agreement property of consensus, p_2 and p_3 are forced to decide 1; while in the state $I \cdot \pi_{\overline{3}-1}$, p_2 and p_3 cannot see the input value of p_1 and this implies that p_2, p_3 decide a unique element u of the set $\{2, 3\}$ and p_1 also decides u. As we have said before, p_2 and p_3 cannot distinguish between the states $I \cdot \pi_1$ and $I \cdot \pi_*$ and they decide 1 in the state $I \cdot \pi_1$, then they also decide 1 in $I \cdot \pi_*$. On the other hand, p_1 cannot distinguish between $I \cdot \pi_*$ and $I \cdot \pi_{\overline{3}-1}$ and in the later state p_1 decides u, gather that, p_1 also decides u in $I \cdot \pi_*$. We conclude that in the execution in which $\mathcal{A}$ enters the state $I \cdot \pi_*$, p_2 and p_3 decide 1 but p_1 decides $u \neq 1$. This constitutes a violation of the Agreement condition of the consensus problem, then the protocol cannot end in one iteration. If the processes execute $\mathcal{A}$ for one more round and decide, then because we work in an iterated model, we can prove that $\mathcal{S}_{\mathcal{A}}$ will contain the subcomplex given at the bottom of Figure 6; it has simplexes representing reachable states in the second round of $\mathcal{A}$ of the form

$$(I \cdot \rho) \cdot \sigma \qquad \rho, \sigma \in \{\pi_1, \pi_*, \pi_{\overline{3}-1}\},$$

and it can be verified that this complex contains a sequence of states that connect $(I \cdot \pi_1) \cdot \pi_1$ with $(I \cdot \pi_{\overline{3}-1}) \cdot \pi_{\overline{3}-1}$ and this fact can be used to show that there is an execution of the second round of $\mathcal{A}$ in which processes decide two different values, again violating the Agreement property of consensus. As we work in an iterated model, we can repeat the same argument in every round of $\mathcal{A}$ and that would imply that $\mathcal{A}$ cannot solve the consensus problem for three processes.

The formal results

We now formalize the main ideas of the example above to prove that it is impossible to solve n-consensus ($n \geqslant 3$) in the IFSC model.

Lemma 3.3 *Let $n \geqslant 3, i \in \overline{n}$. If $\mathcal{A}$ is a protocol in the IFSC model and S is any reachable state in some round $r \geqslant 0$ of $\mathcal{A}$, then there exists the sequence*

$$S \cdot \pi_i \overset{\overline{n}-i}{\sim} S \cdot \pi_* \overset{i}{\sim} S \cdot \pi_{\overline{n}-i}, \tag{3.1}$$

of connected reachable states in round $r+1$ of $\mathcal{A}$. Moreover, the value of the safe-consensus object is the same in all three states.

Proof Let S be a reachable state in $\mathcal{A}$ and $i \in \overline{n}$. We first show that there exists executions of $\mathcal{A}$ in which the return value of the safe-consensus object is the same in the three states $S \cdot \pi_i, S \cdot \pi_*$ and $S \cdot \pi_{\overline{n}-i}$ respectively. If this fact is true, then it will be clear that the given states can be connected in the way described by (3.1).

In the state $S \cdot \pi_i$, the value of the safe-consensus object is i, because p_i executes the shared object before any other process executes it and by the Validity condition of the safe-consensus task, the return value must be the value proposed by p_i. For the states $S \cdot \pi_*$ and $S \cdot \pi_{\overline{n}-i}$, the value of the safe-consensus can be arbitrary. This is true because in each round, all processes invoke the same safe-consensus object and by hypothesis $n \geqslant 3$ (this implies that $|\overline{n}|, |\overline{n} - i| \geqslant 2$). Thus in the executions of $\mathcal{A}$ (given by the round schedules π_* and $\pi_{\overline{n}-i}$) that take the protocol into the states $S \cdot \pi_*$ and $S \cdot \pi_{\overline{n}-i}$, at least two processes execute the safe-consensus object concurrently. By the Validity property, the return value of the object can be arbitrary. Therefore there exists the possibility that the value returned by the safe-consensus object to all processes is precisely i in the states $S \cdot \pi_i, S \cdot \pi_*$ and $S \cdot \pi_{\overline{n}-i}$. It follows that the sequence (3.1) of connected reachable states in $\mathcal{A}$ exists. □

Lemma 3.4 *Let $n \geqslant 3, i \in \overline{n}$. If $\mathcal{A}$ is a protocol in the IFSC model and there exists a sequence*

$$S_0 \overset{X_1}{\sim} \cdots \overset{X_l}{\sim} S_l \qquad (l \geqslant 1)$$

of connected reachable states in $\mathcal{A}$ such that for all j with $1 \leqslant j \leqslant l$, $X_j = \overline{n} - i$ or $X_j = \{i\}$. Then there exists a sequence

$$Q_0 \overset{Z_1}{\sim} \cdots \overset{Z_s}{\sim} Q_s \qquad (s \geqslant 1)$$

of connected reachable states in $\mathcal{A}$ such that

(1) Every state Q_t is a successor state of some S_j.

(2) For all m with $1 \leqslant m \leqslant s$, Z_m is such that $Z_m = \overline{n} - i$ or $Z_m = \{i\}$.

Proof We use induction on l, the round schedules $\pi_i, \pi_*, \pi_{\overline{n}-i}$ and Lemma 3.3 to construct the sequence $Q_1, \ldots, Q_s$ of connected states with the desired properties. For the base case, consider the states $S_0 \overset{X_1}{\sim} S_1$ where $X_1 = \{i\} \vee X_1 = \overline{n} - i$. It is easy to see that the state $S_0 \cdot \rho$ is adjacent to the state $S_1 \cdot \rho$, where ρ is π_i if $X_1 = \{i\}$ and is $\pi_{\overline{n}-i}$ if $X_1 = \overline{n} - i$. Assume that we have build the sequence

$$Q_0 \overset{Z_1}{\sim} \cdots \overset{Z_{s'}}{\sim} Q_{s'},$$

of connected successor states of $S_1, \ldots, S_q$ $(1 \leqslant q < l)$ with $Z_m = \overline{n} - i$ or $Z_m = \{i\}$ for all $m \leqslant s'$. Each state Q_m is of the form $Q_m = S_j \cdot \alpha$, where $\alpha \in \{\pi_i, \pi_*, \pi_{\overline{n}-i}\}$. We now show how to connect $Q_{s'}$ (a successor state of S_q) with a successor state of S_{q+1}. Let X_{q+1} be the set of processes that cannot distinguish between S_q and S_{q+1}. We have to deal with cases.

Case 1. $Q_{s'} = S_q \cdot \pi_i$. If $X_{q+1} = \{i\}$, then we can connect $S_q \cdot \pi_i$ with $S_{q+1} \cdot \pi_i$ (p_i cannot distinguish between these two states). Now, if $X_{q+1} = \overline{n} - i$, using Lemma 3.3 we can build the sequence

$$S_q \cdot \pi_i \overset{\overline{n}-i}{\sim} S_q \cdot \pi_* \overset{i}{\sim} S_q \cdot \pi_{\overline{n}-i} \overset{\overline{n}-i}{\sim} S_{q+1} \cdot \pi_{\overline{n}-i}.$$

Case 2. $Q_{s'} = S_q \cdot \pi_*$. Whether $X_{q+1} = \{i\}$ or $X_{q+1} = \overline{n} - i$ we have the sequence of connected states

$$S_q \cdot \pi_* \overset{X_{q+1}}{\sim} S_q \cdot \pi_{X_{q+1}} \overset{X_{q+1}}{\sim} S_{q+1} \cdot \pi_{X_{q+1}}.$$

Case 3. $Q_{s'} = S_q \cdot \pi_{\overline{n}-i}$. The argument here is very similar to Case 1, so we omit it.

We have build with induction the sequence of connected states $Q_1, \ldots, Q_s$ from the sequence $S_1, \ldots, S_l$ satisfying the demanded properties. The result follows. □

Lemma 3.5 *Let $n \geqslant 3, i \in \overline{n}$. Suppose that $\mathcal{A}$ is a protocol in the IFSC model that solves the consensus problem and that $S_0 \overset{X_1}{\sim} \cdots \overset{X_l}{\sim} S_l$ $(l \geqslant 1)$ is a sequence of connected reachable states in $\mathcal{A}$ that satisfies the hypothesis of Lemma 3.4 and also assume that S_0 is a v-valent state. Then S_l is v-valent.*

Proof It is enough to prove the lemma for $l = 1$, as the general case follows easily from this case. Suppose (without loss of generality) that $S = S_0$ is v-valent and that $S' = S_l$ is v'-valent ($v \neq v'$). Since S is v-valent, in every execution starting from S, there is at least one process that outputs v and by the Agreement condition of consensus, all processes must output v as the consensus value; the same is true for S', replacing v with v'. Let r be the round number of both S and S' (this makes sense, because S and S' are adjacent, thus they must have the same round number, which is part of the local state of each process), then combining Lemma 3.4 with an inductive argument, we can find for all $m \geqslant r$ and any two r-round partial executions

$$R_0, \pi_1, \ldots, \pi_r, R_r = S \quad \text{and} \quad P_0, \pi'_1, \ldots, \pi'_r, P_r = S',$$

that end in S and S' respectively, m-round partial executions

$$R_0, \pi_1, \ldots, \pi_r, R_r, \pi_{r+1}, \ldots, \pi_m, R_m \quad \text{and} \quad P_0, \pi'_1, \ldots, \pi'_r, P_r, \pi'_{r+1}, \ldots, \pi'_m, P_m,$$

such that R_m and P_m are adjacent states for all $m \geqslant r$. As the protocol solves the consensus problem, there must exists a k such that R_u and P_u are decision states for all $u \geqslant k$. Without loss, we can assume that $k \geqslant r$. Since R_u is a successor state of S, which is a v-valent state, all processes decide v in R_u; simillary, as P_u is a successor of S' (a v'-valent state), processes must decide v' in P_u. Let X be the set of ids of processes that cannot distinguish between R_u and P_u. Then, for each $j \in X$ the local state of process p_j in P_u and R_u must be the same and this includes its output component. But this is a contradiction because in R_u, p_j decides v while in the state P_u, p_j decides v'. We conclude that S' must be v-valent, such as S. □

Theorem 3.6 *For $n \geqslant 3$, there is no protocol in the IFSC model to solve consensus for n processes.*

Proof Assume that there exists a protocol $\mathcal{A}$ for consensus in the IFSC model and (without loss of generality) suppose that 0,1 are two valid input values. Let i be any process id and let O, U be the initial states in which all processes have as input values 0s and 1s respectively. Clearly, O is a 0-valent state and U is a 1-valent state. Let OU be the initial state in which p_i has input value 0 and all other processes have input value 1. Then in the first round of $\mathcal{A}$, we have the following sequence of connected reachable states in $\mathcal{A}$:

$$O \cdot \pi_i \overset{i}{\sim} OU \cdot \pi_i \overset{\overline{n}-i}{\sim} OU \cdot \pi_* \overset{i}{\sim} OU \cdot \pi_{\overline{n}-i} \overset{\overline{n}-i}{\sim} U \cdot \pi_{\overline{n}-i}.$$

Because $O \cdot \pi_i$ is 0-valent, Lemma 3.5 tell us that the state $U \cdot \pi_{\overline{n}-i}$ is also 0-valent. But this contradicts the fact that $U \cdot \pi_{\overline{n}-i}$ is a 1-valent state. Therefore no such protocol $\mathcal{A}$ can exists. □

4 Conclusion

We have described the iterated model of distributed computing, a useful tool to study and understand the behavior of distributed systems. Also, we described how the runs of protocol in this model can be represented with simplicial complexes and we presented some standard tools (i.e. connectivity of states, valency, etc.) commonly used to investigate distributed systems. Also, we described the set agreement task, one of the most important distributed problems.

To show to the reader what sort of results can be achieved with the iterated model and the topological approach, we defined the iterated shared memory with full safe-consensus objects (IFSC) model, an extension of the basic iterated model using the safe-consensus task of [2]. We showed how to analyze protocol complexes in the IFSC model, and we discovered that the protocol complexes of protocols in the IFSC model are (globally) disconnected. We explained why local connectivity between some pairs of vertices of the protocol complex is sufficient to prove that consensus is unsolvable in the IFSC model. Thus we see that connectivity of the protocol complex (which is related to the notion of non-distinguishable states) is indeed fundamental in the study of distributed computing.

But we also proved that in the new IFSC model we can solve $(n, n-1)$-set agreement. As this problem cannot be solved in the basic iterated model [8,23,18], we can conclude that the IFSC model is indeed more powerful that the basic iterated model.

Several questions remain open. For example, is there an iterated model extended with safe-consensus objects that is equivalent (for task solvability) to the wait-free standard model extended with safe-consensus objects?

References

[1] Afek, Y., H. Attiya, D. Dolev, E. Gafni, M. Merritt and N. Shavit, *Atomic snapshots of shared memory*, J. ACM **40** (1993), pp. 873–890.

[2] Afek, Y., E. Gafni and O. Lieber, *Tight group renaming on groups of size g is equivalent to g-consensus*, in: *Proceedings of the 23rd international conference on Distributed computing (DISC'09)*, LNCS **5805** (2009), pp. 111–126.

[3] Afek, Y., I. Gamzu, I. Levy, M. Merritt and G. Taubenfeld, *Group renaming*, in: *OPODIS '08: Proceedings of the 12th International Conference on Principles of Distributed Systems*, LNCS **5401** (2008), pp. 58–72.

[4] Alexandrov, P. S., "Combinatorial topology. Vol. 1, 2 and 3," Dover Publications Inc., Mineola, NY, 1998, 650 pp., translated from the Russian, Reprint of the 1956, 1957 and 1960 translations.

[5] Armstrong, M. A., "Basic Topology," Springer-Verlag, 1983.

[6] Attiya, H. and S. Rajsbaum, *The combinatorial structure of wait-free solvable tasks*, SIAM J. Comput. **31** (2002), pp. 1286–1313.

[7] Biran, O., S. Moran and S. Zaks, *A combinatorial characterization of the distributed 1-solvable tasks*, J. Algorithms **11** (1990), pp. 420–440.

[8] Borowsky, E. and E. Gafni, *Generalized flp impossibility result for t-resilient asynchronous computations*, in: *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (1993), pp. 91–100.

[9] Borowsky, E. and E. Gafni, *Immediate atomic snapshots and fast renaming*, in: *PODC '93: Proceedings of the twelfth annual ACM symposium on Principles of distributed computing* (1993), pp. 41–51.

[10] Borowsky, E. and E. Gafni, *A simple algorithmically reasoned characterization of wait-free computation (extended abstract)*, in: *PODC '97: Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing* (1997), pp. 189–198.

[11] Chaudhuri, S., *More choices allow more faults: set consensus problems in totally asynchronous systems*, Inf. Comput. **105** (1993), pp. 132–158.

[12] Conde, R. and S. Rajsbaum, *Two normal forms of an iterated snapshot model and their power to solve consensus from safe-consensus* (2010), manuscript.

[13] Fischer, M. J., N. A. Lynch and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. ACM **32** (1985), pp. 374–382.

[14] Gafni, E. and S. Rajsbaum, *Musical benches*, in: *Proceedings of the 19th international conference on Distributed computing (DISC'05)*, LNCS **3724** (2005), pp. 63–77.

[15] Gafni, E. and S. Rajsbaum, *Distributed programming with tasks*, in: *OPODIS '10: Proceedings of the 14th International Conference on Principles of Distributed Systems*, Lecture Notes in Computer Science **6490** (2010), pp. 205–218.

[16] Gafni, E., S. Rajsbaum and M. Herlihy, *Subconsensus tasks: Renaming is weaker than set agreement*, in: *Proceedings of the 20th international conference on Distributed computing (DISC'06)*, LNCS **4167** (2006), pp. 329–338.

[17] Herlihy, M. and S. Rajsbaum, *A classification of wait-free loop agreement tasks*, Theoretical Computer Science **291** (2003), pp. 55 – 77.

[18] Herlihy, M. and N. Shavit, *The asynchronous computability theorem for t-resilient tasks*, in: *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (1993), pp. 111–120.

[19] Herlihy, M. and N. Shavit, *The topological structure of asynchronous computability*, J. ACM **46** (1999), pp. 858–923.

[20] Loui, M. C. and H. H. Abu-Amara, *Memory requirements for agreement among unreliable asynchronous processes*, Parallel and Distributed Computing, Advances in Computing Research. F. P. Preparata ed., JAI Press, Greenwich, CT **4** (1987), pp. 163–183.

[21] Rajsbaum, S., *Iterated shared memory models*, in: *LATIN '2010: Proceedings of the 9th Latin American Symposium on Theoretical Informatics*, Lecture Notes in Computer Science **6034** (2010), pp. 407–416.

[22] Rajsbaum, S., M. Raynal and C. Travers, *The iterated restricted immediate snapshot model*, in: *COCOON '08: Proceedings of the 14th annual international conference on Computing and Combinatorics* (2008), pp. 487–497.

[23] Saks, M. and F. Zaharoglou, *Wait-free k-set agreement is impossible: the topology of public knowledge*, in: *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (1993), pp. 101–110.

[24] Saks, M. and F. Zaharoglou, *Wait-free k-set agreement is impossible: The topology of public knowledge*, SIAM J. Comput. **29** (2000), pp. 1449–1483.