Computing in the Presence of Concurrent Solo Executions

Maurice Herlihy[1], Sergio Rajsbaum[2], Michel Raynal[3,4], and Julien Stainer[4]

[1] Brown University, Providence (RI), USA
[2] Instituto de Mathematicas, UNAM, D.F. 04510, Mexico
[3] Institut Universitaire de France
[4] IRISA Université de Rennes 1, INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract. In a *wait-free* model any number of processes may crash. A process runs *solo* when it computes its local output without receiving any information from other processes, either because they crashed or they are too slow. While in wait-free shared-memory models at most one process may run solo in an execution, any number of processes may have to run solo in an asynchronous wait-free message-passing model.

This paper is on the computability power of models in which several processes may concurrently run solo. It first introduces a family of round-based wait-free models, called the *d-solo* models, $1 \leq d \leq n$, where up to d processes may run solo. The paper gives then a characterization of the colorless tasks that can be solved in each *d-solo* model. It also introduces the (d, ϵ)*-solo approximate agreement* task, which generalizes ϵ-approximate agreement, and proves that (d, ϵ)-solo approximate agreement can be solved in the d-solo model, but cannot be solved in the $(d + 1)$-solo model. The paper studies also the relation linking d-set agreement and (d, ϵ)-solo approximate agreement in asynchronous wait-free message-passing systems.

These results establish for the first time a hierarchy of wait-free models that, while weaker than the basic read/write model, are nevertheless strong enough to solve non-trivial tasks.

1 Introduction

Distributed computability. The computability power of a distributed model depends on its communication, timing, and failure assumptions. A basic result is the impossibility to solve consensus in an asynchronous read/write [16] or message-passing [8] system even if only one process may crash. When looking at the communication medium and assuming asynchronous processes prone to crash failures, a read/write system and a message-passing system have the same computability power if and only if less than half of the processes may crash [1]. If a majority of the processes may crash, the message passing model is weaker than the shared memory model because partitions can occur.

The power of a distributed model has been studied in detail with respect to *tasks*, which are the distributed equivalent of functions in sequential computing. Each process gets only one part of the input, and after communicating with the others, decides on an output value, such that collectively, the various local outputs produced by the processes respect the task specification, which is defined from the local inputs of the processes. This paper concentrates on the class of *colorless tasks* (e.g., [3,12]), where the specification is in terms of possible inputs and outputs, but without referring to which

© Springer-Verlag Berlin Heidelberg 2014

process gets which input or produces which output. Among the previously studied notable tasks, many are colorless, such as consensus [8], set agreement [5], approximate agreement [6], loop agreement [13] while some are not, like renaming [2].

Wait-freedom and solo execution. This paper considers *wait-free* distributed crash-prone asynchronous models. Wait-free has two (complementary) meanings. First, it means that the model allows up to $n - 1$ processes to crash, where n is total number of processes. Its other meaning expresses a liveness condition, namely it requires that every non-faulty process progresses and eventually decides (i.e., computes a result) whatever the behavior of the other processes [11].

In a wait-free model where processes must satisfy the wait-freedom liveness condition, a process has to make progress even in the extreme cases where all other processes have crashed, or are too slow, and consequently be forced to decide without knowing their input values. Hence, for each process, there are executions where this process perceives itself as being the only process participating in the computation.

More generally, we say that a process executes *solo* if it computes its local output without knowing the input values of the other processes.

Two extreme wait-free models: shared memory and message passing. In a model where processes communicate by reading and writing shared registers, at most one process can run solo in any execution. This is because, when a process runs solo, it writes and reads from the shared memory, and eventually writes its decision. Any other process that starts running, will be able to read the history left by the solo process in the memory.

When considering message-passing communication, all processes may have to run solo concurrently in the extreme case, where messages are arbitrarily delayed, and each process perceives the other processes as having crashed. Only tasks that can be solved without communication can be computed in this model.

Investigating the computability power of intermediary models. The aim of the paper is to study the computability power of asynchronous models in which processes may run solo in the same execution. More precisely, assuming that up to d processes may run solo, the paper addresses the following questions:

- How to define a computation model in which up to d processes may run solo?
- Which tasks can be computed in such a model?

The aim is to study these questions in a clean theoretical framework, and (for the first time) investigate models weaker than the basic wait-free read/write model. However, we hope that our results are relevant to other intermediate models, such as distributed models over fixed or wireless networks.

To simplify the technical development, following [4], the paper develops a theoretical round-based framework, iterated model (IIS) that has been proved useful in many other papers. Processes execute an infinite sequence of asynchronous rounds and communicate through specific objects called *immediate snapshot* objects. Such objects are high-level read/write objects such that a new object instance is associated with each round and, when it executes a round r, a process can access only the object associated with round r. A main interest of the IIS model is that, from a task computability point of view, it has the same power as the read/write wait-free model [4]. Also, the topology of the IIS model is easier to analyze, establishing a good foundation to analyze task solvability in various distributed computing models.

Contributions. The following contributions answer previous questions:

- The definition of a family of *d-solo models,* each parametrized with an integer d, $1 \leq d \leq n$. The 1-solo model corresponds to the IIS model (which is equivalent to the read/write wait-free model [4]), while the n-solo model corresponds to the round-based wait-free message-passing model.
- A characterization of the set of colorless tasks that can be solved in the d-solo model, $1 \leq d \leq n$. Via a new form of complex subdivisions, this characterization connects topology with *colorless* algorithms.
- Any *d-solo* model with $d \geq 2$, is weaker than the read/write wait-free model, yet there are natural, non-trivial tasks that can be solved in the *d-solo* model. One of these tasks, called *(d, ϵ)-solo approximate agreement* (in short (d, ϵ)-SAA) is such that (d, ϵ)-SAA can be solved in the d-solo model, for any $\epsilon > 0$, but not in the $(d + 1)$-solo model. Hence, more tasks can be solved in the d-solo model than in the $(d + 1)$-solo model, for $1 \leq d < n$, which establishes a hierarchy of solo models.
- Finally, the d-solo model is related to d-set agreement. This relation shows that, for $d < n$, d-set agreement is strong enough to solve (d, ϵ)-solo approximate agreement but is too weak to solve $(d-1, \epsilon)$-solo approximate agreement in the wait-free message-passing model. This provides us with a better insight on a bound on the "maximal partitioning" allowed to solve $(d - 1, \epsilon)$-solo approximate agreement in the wait-free message-passing model.

The (d, ϵ)-solo approximate agreement task is a generalization of approximate agreement [6]. The input of each process consists of a point in the Euclidean space $\mathbb{R}^N$ ($N \geq d$). The *validity* property states that each process p_i has to decide a point which is in the convex hull of all the input points. The *agreement* property states that at most d processes may decide any point in the convex hull of the input points (let CH be the convex hull defined by these at most d points), while the other processes have to decide values whose distance to CH is at most ϵ. Actually, the convex hull of solo processes is an "attractor" for the set of decided values.

When $d = 1$, validity and agreement imply that the Euclidean distance between any pair of points decided by the processes has to be upper bounded by a predefined constant. Thus, $(1, \epsilon)$-solo approximate agreement problem in $\mathbb{R}^m$ is essentially the problem that has been recently considered in the context of t Byzantine failures and asynchronous message-passing systems [17,20], where it is shown that it can be solved iff $n > t(m + 2)$.

The colorless tasks that are solvable in the wait-free iterated immediate snapshot (IIS) model have been characterized in [12]. Due to the simulations in [4,10], this characterization holds for the usual read/write wait-free model. Section 4 extends the characterization of [12] to the d-solo model, $1 \leq d \leq n$. Our characterization in terms of colorless algorithms permits the use of standard subdivisions, instead of chromatic subdivisions used in previous papers. We believe colorless algorithms are interesting in themselves, and indeed, for $d = 1$, if a colorless task is solvable, it is solvable by a colorless algorithm. For $d > 1$ we defer the proof that colorless algorithms and general algorithms can solve a very similar class of tasks.

One of the central results of topology is the Simplicial Approximation Theorem [18], which establishes what is a "discrete version" of a continuous map. This theorem is also central for the wait-free characterization theorem of [15] and its t-resilient extension (e.g., [12]). However, this theorem cannot be used in a d-solo model, $d > 1$, because it is no longer the case that the diameter of the simplexes in a subdivision is reduced. Not even the Relative Simplicial Approximation Theorem [21] can be directly used.

Finally, it is important to notice that our d-solo model addresses different issues than the d-concurrency model of [9], where it is shown that with d-set agreement any number of processes can emulate d state machines of which at least one remains highly available. While d-concurrency is used to reduce the concurrency degree to at most d processes that are always allowed to cooperate, d-solo allows up to d processes to run independently (i.e., without any cooperation).

Roadmap. The paper is composed of 6 sections. Section 2 introduces base definitions, the communication objects, and the d-solo model. Section 3 investigates colorless tasks in the d-solo model, while Section 4 focuses on what can be computed in the presence of concurrent solo executions. Then, Section 5 defines the (d, ϵ)-solo approximate agreement problem, shows that it can be solved in the d-solo model and cannot in the $(d+1)$-solo model, thereby defining a strict hierarchy of distributed computing models. Section 6 concludes the paper. Due to page limitation, topology notions, all proofs, additional technical developments, and relations between d-set agreement and (d, ϵ)-solo approximate agreement in wait-free message-passing systems are given in [14].

2 Tasks, Processes, Communication Object, and Iterated Model

Tasks. A task is a one-shot distributed computing problem specified in terms of an input/output relation Δ. Each process starts with a private input value and must eventually compute a private output value. The task specifies the possible initial configurations. An initial configuration I specifies the input value of each process. Similarly, the output values produced by the processes in an execution represents an output configuration O.

A task $(\mathcal{I}, \mathcal{O}, \Delta)$ is defined by a set of input configurations $\mathcal{I}$, a set of possible output configurations $\mathcal{O}$, and a relation Δ which specifies which output configurations $O \in \mathcal{O}$ are correct for each input $I \in \mathcal{I}$. A more formal description appears in Section 3.1 and in previous papers such as in [15].

Processes. The system model is made up of n asynchronous (deterministic) sequential processes, $p_1, \ldots, p_n$, which proceed in asynchronous rounds [19]. The index i of process p_i is sometimes used to denote p_i. Up to $n - 1$ processes may crash. Once a process crashes, it never recovers. We say the model is *wait-free*.

Rounds and communication objects. A communication object $CO[r]$ is associated with each round r and this object is the only means for the processes to communicate during round r. The rounds are *communication-closed* [7] which means that, when a process executes a round, it can communicate with other processes only through the object associated with this round.

More precisely, $CO[r]$ is a one-shot object (i.e., each process accesses it only once) which provides the processes with a single operation denoted $\mathsf{communicate}(i, v)$, where v is the value that the invoking process p_i wants to communicate to the other processes during round r. Such an invocation returns to p_i a set of pairs (process identity, value) deposited into $CO[r]$ by other processes during round r.

Iterated model. Each process p_i executes the algorithm skeleton described in Figure 1, where the local computation parts are related to the particular task that is solved. The local variable r_i is the local round number, ℓs_i contains p_i's local state, while $view_i$ contains all the pairs $(j, \ell s_j)$ communicated to p_i during the current round. The local transition function $\delta_i()$ defines the new local state of p_i according to its previous local state and the pairs $(j, \ell s_j)$ it has obtained from $CO^d[r]$ (the parameter d is explained below in Section 2.1). To solve a task, it is necessary to instantiate accordingly $\delta_i()$, the predicate decision() and the function dec_val(): decision() allows p_i to decide, while dec_val() allows it to compute the decided value. As we are interested in computability and not efficiency, we assume a *full information* algorithm, i.e., at the end of each round r_i, ℓs_i contains the value of $view_i$, and δ_i can be task independent. However, we will see in Section 3 that in some cases, tasks can be solved without communicating all a process knows.

```
(01)  $r_i \leftarrow 0$; $\ell s_i \leftarrow$ initial local state;
(02)  **loop forever** $r_i \leftarrow r_i + 1$; $view_i \leftarrow CO^d[r_i]$.communicate$(i, \ell s_i)$;
(03)                   $\ell s_i \leftarrow \delta_i(\ell s_i, view_i)$; **if** decision$(\ell s_i)$ **then** dec_val$(\ell s_i)$ **end if**
(04)  **end loop**.
```

Fig. 1. Generic iterated model

2.1 Communication Object

The communication objects $CO^d[1]$, $CO^d[2]$, etc., of an execution are parametrized by a solo-dimension d, $1 \leq d \leq n$. As previously indicated, an object $CO^d[r]$ contains a set of pairs, one per process. Each pair (i, v) is such that i is a process index and v the value communicated by p_i, and $CO^d[r]$ contains at most one pair per process.

Definition. The behavior of every object CO^d is defined as follows. Considering an execution during which each of the n processes $\{p_1, \ldots, p_n\}$ accesses the object (at most once) using its local state ℓs_i as input, one can represent this execution by an *ordered partition*, i.e., a tuple of non-empty sets $(P_1, \ldots, P_z)$ such that (1) for any distinct $i, j \in \{1, \ldots, z\}$: $P_i \cap P_j = \emptyset$, and (2) $\bigcup_{i=1}^{z} P_i = \{p_1, \ldots, p_n\}$. From an operational view, the ordered partition $(P_1, \ldots, P_z)$ describes the sequence of concurrent accesses to the object CO^d.

The behavior of CO^d is defined from a *d-ordered partition*, where a d-ordered partition is an ordered partition $(\pi_1, \ldots, \pi_{z'})$ such that $0 \leq |\pi_1| \leq d$ (the size of the first set of the partition can be 0 and cannot exceed d). More precisely, the d-ordered partition $(\pi_1, \ldots, \pi_{z'})$ associated with CO^d is:

- If $|P_1| > d$: $(\pi_1, \ldots, \pi_{z'}) = (\emptyset, P_1, \ldots, P_z)$, and
- If $|P_1| \leq d$: $(\pi_1, \ldots, \pi_{z'}) \in \{(\emptyset, P_1, \ldots, P_z), (P_1, \ldots, P_z)\}$.

$(\pi_1, \ldots, \pi_{z'}) = (P_1, \ldots, P_z)$ captures the cases where, initially, d (or less) processes execute solo. In the other cases we have $(\pi_1, \ldots, \pi_{z'}) = (\emptyset, P_1, \ldots, P_z)$, because initially either too many processes execute concurrently (first item), or, while no more than d processes execute concurrently, none of them executes solo.

The values $view_i$, $1 \leq i \leq n$, obtained by the processes when the behavior of CO^d is represented by the *d-ordered partition* $(\pi_1, \ldots, \pi_{z'})$ are defined as follows:

$$(i \in \pi_1) \Rightarrow (view_i = \{(i, \ell s_i)\}), \quad \text{and}$$
$$(x > 1 \wedge i \in \pi_x) \Rightarrow \big(view_i = \{(j, \ell s_j) \ : \ j \in \pi_y \wedge y \leq x\}\big).$$

This means that the view of each process p_i belonging to π_1 (where $0 \leq |\pi_1| \leq d$) contains only its own contribution, namely the pair (i, ls_i). Differently, the view of a process p_i in π_x, where $x > 1$, contains all the pairs $(j, \ell s_j)$ deposited in CO^d by the processes p_j of the sets π_y such that $y \leq x$. Thus, each process of π_1 appears as executing solo, while each other process of a set p_x, $x \neq 1$, sees the contributions provided (a) by all the processes p_i belonging to the "previous" sets π_y $(y < x)$, and (b) by all the processes from its "concurrency" set π_x. (The immediate snapshot object described in [3] implements CO^d for $d = 1$.) Examples of communication objects are presented in [14].

Object properties. Given an object CO^d, the next properties follows from its definition (See examples of CO^d objects in the Appendix).

- Solo execution upper bound. $0 \leq |\{i \text{ such that } |view_i| = 1\}| \leq d$.
- Self-inclusion. $\forall\, i : (i, -) \in view_i$.
- Containment. $\forall\, i, j : \big((|view_i| \leq |view_j|) \wedge |view_j| > 1)\big) \Rightarrow (view_i \subseteq view_j)$.

2.2 A Spectrum of Solo Models

It follows from their definition that CO^d is stronger (more constraining) than CO^{d+1} in the sense that the subdivided complex of CO^d is included the one of C^{d+1}. Intuitively, this means that CO^d includes "more synchrony" than CO^{d+1}.

The d-solo model. The generic framework described in Figure 1 instantiated with CO^d objects is called the *d-solo model*. It is denoted $\mathcal{ACS}^d_{n,n-1}$(ASC stands for Asynchronous Concurrent Solo) where the first subscript denotes the total number of processes, while the second subscript denotes the upper bound on the number of processes allowed to crash.

Hierarchy of d-solo models. Let $A \succeq_T B$ mean that any task that can be solved in the model B can be solved in the model A, and $A \simeq_T B \overset{def}{=} (A \succeq_T B) \wedge (B \succeq_T A)$.

Let $\mathcal{ARW}_{n,n-1}$ denote the base wait-free (asynchronous) read/write model. It follows from the fact that (for task solvability) the IIS model and $\mathcal{ARW}_{n,n-1}$ have the same computability power [4], and IIS is nothing more than $\mathcal{ACS}^1_{n,n-1}$, that we have $\mathcal{ARW}_{n,n-1} \simeq_T \mathcal{ACS}^1_{n,n-1}$.

Let $\mathcal{AMP}_{n,n-1}$ denote the classical (non-iterated) message-passing system where up to $(n-1)$ processes may crash. As all processes except one may crash and communication is asynchronous (hence messages can be arbitrarily delayed), the tasks that can be solved in $\mathcal{AMP}_{n,n-1}$ are the tasks that can be wait-free solved without communication. But, this set of tasks is exactly the set of tasks that can be solved in $\mathcal{ACS}^n_{n,n-1}$. Hence, $\mathcal{ACS}^n_{n,n-1} \simeq_T \mathcal{AMP}_{n,n-1}$.

It follows from the definition of the communication objects CO^d and CO^{d+1} that any task solvable in $\mathcal{ACS}^{d+1}_{n,n-1}$ is solvable in $\mathcal{ACS}^d_{n,n-1}$. We have consequently the following hierarchy of models: $\mathcal{ARW}_{n,n-1} \simeq_T \mathcal{ACS}^1_{n,n-1} \succeq_T \cdots \succeq_T \mathcal{ACS}^d_{n,n-1} \succeq_T \ldots \succeq_T \mathcal{ACS}^n_{n,n-1} \simeq_T \mathcal{AMP}_{n,n-1}$. We will see in Section 5 that $A \succeq_T B$ can be replaced by $A \succ_T B$ (all the tasks solvable in B are solvable in A, and there is one task solvable in A and not in B).

3 Colorless Tasks and the d-Solo Model

This section focuses on colorless tasks that can be solved in the d-solo model. After having defined colorless tasks it shows that, for these tasks, one can use a restricted form of the algorithm in Figure 1. It then, introduces the notions of a (d, R)-subdivision task and a (d, R)-agreement task. (More topology notions are given in [14].)

3.1 Colorless Tasks

A colorless task is a special kind of task where the processes cannot use their ids during the computation. This implies that the task specification is not in terms of ids. A colorless task specifies which sets of values are valid input configurations, and which are valid output decisions, but not which value is assigned to which process. Thus, a process may adopt the input value or the output value of another process.

Formally, a *colorless task* is a triple $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, where $\mathcal{I}^*$ is a colorless *input complex*, $\mathcal{O}^*$ is a colorless *output complex*, and $\Delta^* : \mathcal{I}^* \rightarrow 2^{\mathcal{O}^*}$ is a *carrier map*. A colorless complex is a family of sets, over some basic set of values, such that if a set is in the complex, then all its subsets are also in the complex. A set in the complex is called a *simplex*. Simplexes of size 1, are called *vertices*, and of size 2, *edges*. Indeed, a graph is a 1-dimensional complex. In the case of a colorless complex, a vertex is just a value, either an input or an output value, while in a colored complex, a vertex is a pair of values, one is a process id, and the other is an input our output value. If σ is an input simplex in $\mathcal{I}^*$, the carrier map $\Delta^*(\sigma)$ is a subcomplex of $\mathcal{O}^*$ satisfying *monotonicity:* $\forall \sigma, \sigma' \in \mathcal{I}^* \ : \ \Delta^*(\sigma \cap \sigma') \subseteq \Delta^*(\sigma) \cap \Delta^*(\sigma')$.

Operationally, the meaning of a colorless task is the following. If $\sigma \in \mathcal{I}^*$, then the processes can start an execution with input values from σ; different processes may propose the same vertex or different vertices from σ. Processes eventually decide (not necessarily distinct) vertices that belong to the same output simplex $\tau \in \mathcal{O}^*$, such that $\tau \in \Delta^*(\sigma)$. If the system consists of n processes, then the processes can start with at most n different input values, and hence, processes will never start on a simplex σ of $\mathcal{I}$ of dimension greater than $n - 1$ (the dimension of σ is $|\sigma| - 1$). Thus, for n processes, only the simplexes of $\mathcal{I}$ of dimension $\leq n - 1$ are relevant, i.e., the $n - 1$ *skeleton* of $\mathcal{I}$, denoted $\text{Skel}^{n-1}\mathcal{I}$. For example, in a system of two processes, $n = 2$, only the 1-skeleton of $\mathcal{I}$ is of interest, which is the graph consisting of the vertices and 1-simplices of $\mathcal{I}$.

3.2 Colorless Algorithms

A *colorless algorithm* is an algorithm in the form of Figure 1, but where the local computation made by δ_i in line (3) is very restricted. Although a colorless algorithm is not as powerful as an algorithm with no restrictions, it simplifies that exposition, and in the full version we show that they can solve a similar class of colorless tasks.

Informally, in a colorless algorithm processes behave in an anonymous way: processes consider the shared memory as if it is a set. (A colorless complex is denoted with a $*$ superscript, as in $\mathcal{K}^*$). In each round, a process deposits its input in the set, and gets back a view of the contents of the set. If two processes deposit the same value in the set, only one copy is stored. When a process gets back a set of values, there is no information of which process deposited which value. A process "forgets" which is its own value in the set. The set of values that a process receives at the end of a round, becomes its input to the next round.

Formally, in an execution, the initial local state of a process p_i is a vertex v_i of $\mathcal{I}^*$, and is assigned in line 1 to ℓs_i. Furthermore, the set of all initial states v_i (not necessarily distinct) is a simplex σ of $\mathcal{I}^*$. We may write, $\sigma = \{\ell s_1[0], \ldots, \ell s_n[0]\}$, where $\ell s_i[0]$ denotes the initial value of ℓs_i. Notice that $|\sigma|$ may be less than n because different processes may start with the same input value.

The local transition δ_i eliminates process ids. Namely, during any round r and for any process p_i, if we denote by $\ell s_i[r]$ the value of ℓs_i at the end of round r, in line 2 of the algorithm, $view_i$ is assigned the value returned by $CO^d[r].\mathsf{communicate}(i, \ell s_i[r-1])$, and this value is a set of pairs $\{(i_1, \ell s_{i_1}[r-1]), \ldots, (i_k, \ell s_{i_k}[r-1])\}$ that includes ids $i_1, \ldots, i_k$, but when the function δ_i is applied to this set it returns a set $\sigma_i^r = \{\ell s_{i_1}[r-1], \ldots, \ell s_{i_k}[r-1]\}$. We assume every process executes the same number of rounds, $R \geq 0$, and in the last round, produces an output value $\mathsf{dec\_val}(\ell s_i)$ (all processes use the same function $\mathsf{dec\_val}$).

For an R round colorless algorithm in the d-dimensional model, the *algorithm complex* is defined as follows. For each input simplex $\sigma \in \mathcal{I}^*$, the subcomplex $\mathcal{P}^*(\sigma)$ represents the executions r where all processes start with inputs from σ (at least one process starts with each of the vertices in σ). Moreover, in the algorithm complex for the d-dimensional model we do not want to include the $(d-1)$-dimensional model, so we consider only runs where the processes that in a round see more than one process, they see at least $d+1$ processes. The complex $\mathcal{P}^*(\sigma)$ contains a top dimensional simplex $\tau = \{\ell s_i\}$ for each such R round execution of the algorithm starting in σ, where the vertices ℓs_i of τ are the values of $\ell s_i[r]$ at the end of this execution, for each process p_i (without repetitions, as the simplex is a set). The complex $\mathcal{P}^*$ is the union of $\mathcal{P}^*(\sigma)$ over all $\sigma \in \mathcal{I}^*$. It is easy to prove that $\mathcal{P}^*(\cdot)$ is a strict carrier map from $\mathcal{I}^*$ to the algorithm complex $\mathcal{P}^*$.

We will explain the significance of the next lemma later on, when we discuss subdivisions.

Lemma 1. *Consider a 1-round colorless algorithm and an input simplex $\sigma \in \mathcal{I}^*$. The simplexes of $\mathcal{P}^*(\sigma)$ are of the form $\tau = \{\tau_1, \ldots, \tau_z\}$, where each $\tau_i \subseteq \sigma$, and there is an $l, 0 \leq l \leq d$ such that (1) for all $i, 0 \leq i \leq l$, $|\tau_i| = 1$, so $\cup_{0 \leq i \leq l} \tau_i$ is a face σ' of σ, (2) for all $j, l < j \leq z$, $\sigma' \subsetneq \tau_j$, and (3) for all $j, l < j \leq z-1$, $\tau_j \subsetneq \tau_{j+1}$.*

If $\mathcal{P}^*(\cdot)$ is a carrier map from $\mathcal{I}^*$ to the algorithm complex $\mathcal{P}^*$, and $\mathsf{dec\_val}$ is a simplicial map from $\mathcal{P}^*$ to $\mathcal{O}^*$, we say that $\mathsf{dec\_val}$ *is carried by* Δ^* if for each $\sigma \in \mathcal{I}^*$ and each $\tau \in \mathcal{P}^*(\sigma)$, the simplex $\mathsf{dec\_val}(\tau)$ belongs to $\Delta^*(\sigma)$.

Lemma 2. *If the colorless task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$ is solvable by a colorless algorithm then there exists an algorithm complex $\mathcal{P}^*$, and a simplicial map* $\mathsf{dec\_val}$ *from $\mathcal{P}^*$ to $\mathcal{O}^*$ that is carried by Δ^*.*

3.3 (d, R)-Subdivision and (d, R)-Agreement Tasks

The (d, R)-subdivision task. Which is the simplest task a colorless algorithm can solve in the d-dimensional model? It is the task solved when each process executes R rounds, then stops, and its decision function is the identity! Namely, $\mathsf{dec\_val}(\ell s_i) = \ell s_i$ i.e. a process decides the set of values $\ell s_i[R]$ it retrieves from the communication object during the R^{th} round. Given any input complex $\mathcal{I}^*$ and any integer $R \geq 0$, we call this task the (d, R)-*subdivision task* over $\mathcal{I}^*$. The output complex $\mathcal{O}^*$ of this task is of course equal to the algorithm complex $\mathcal{P}^*$, with the simplicial map dec_val being the identity. For the carrier map, $\Delta^*(\sigma)$ includes all simplexes τ that correspond to executions starting in σ, i.e., $\Delta^*(\sigma) = \mathcal{P}^*(\sigma)$. In particular, for $R = 0, \mathcal{I}^* = \mathcal{O}^*$, and Δ^* is the identity carrier map, which sends a simplex σ to the complex consisting of σ and all its faces (which we often denote by σ, abusing notation).

By definition, the (d, R)-subdivision task over $\mathcal{I}^*$ is solvable in the d-dimensional model, and moreover, by a colorless algorithm. In fact, it is the basic building block to solve every other colorless task, as shown in Theorem 1. We will justify the name "subdivision task" when we see how to specify the task without resorting to executions of some model in Section 3.4.

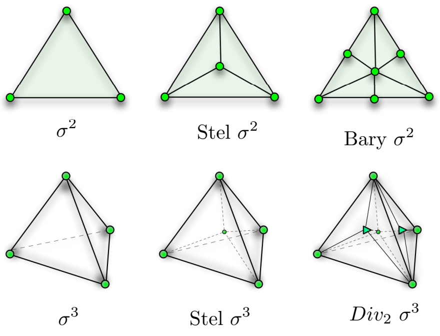
The (d, R)*-agreement task.* When the vertices of $\mathcal{I}^*$ are points in Euclidean space, the (d, R)-subdivision task can be used directly to solve a task that we call (d, R)-*agreement task* over $\mathcal{I}^*$, which is defined combinatorially in Section 5. In the (d, R)-subdivision task, processes propose sets of values in each round. We can encode such a set of values as its barycenter b, and then the process can directly propose b. We shall see in Section 5, that, although both tasks are essentially the same, when we work with barycenters processes compute output values within ϵ of each other (except for at most d processes that may run solo), and we can make ϵ as small as we want, by choosing a large enough value of R.

Operationally, the (d, R)-agreement task over $\mathcal{I}^*$ is defined as follows. Processes execute R rounds of a colorless algorithm in the d-dimensional model. In each round r, each process p_i computes its value $\ell s_i[r]$ that will be the input to the next round, in line 3 of the algorithm, by taking the barycenter of the values that it gets back from the object in line 2. The barycenter computed in round R is the output of of the process.

3.4 The Structure of Colorless Algorithms

The structure of a colorless complex is explained in terms of *subdivisions* (due to page limitation, more developments can be found in [14]). Examples of subdivisions of a simplex are illustrated on the figure that follows at the right of the page.

Perhaps the simplest subdivision is the *stellar* subdivision. Given a complex (abusively denoted σ^m) consisting of an m-simplex $\sigma^m = \{s_0, \ldots, s_m\}$ and all its faces, the complex Stel(σ^m, b) is constructed by taking a *cone* with apex b over the boundary complex $\partial\sigma^m$.



The *barycentric* subdivision, Bary σ^m, is perhaps the most widely used in topology. A simplex τ is in Bary σ^m if and only if there exists a sequence $\sigma_0 \subset \ldots \subset \sigma_z$ of faces of σ^m, and the set of vertices of τ is the set of the barycenters of the these faces, denoted $\hat{\sigma}_i, 0 \leq i \leq z$.

For the d-solo models, we need to define a family of subdivisions that goes from the stellar to the barycentric subdivision. The *d-dimensional subdivision* of a complex $\mathcal{K}$

denoted $\text{Div}_d\ \mathcal{K}$, is the barycentric subdivision of $\mathcal{K}$ relative to $\text{Skel}^{d-1}\mathcal{K}$. Intuitively, we do not subdivide $\text{Skel}^{d-1}\mathcal{K}$ because we consider executions where up to d processes run solo, they get their own view in an invocation of a CO^d object. See the construction of Figure 2 and topology notions in Appendix. As usual, the *R-iterated d-dimensional subdivision*, $\text{Div}_d^R\ \mathcal{K}$, is obtained by repeating the subdivision process R times.

```
(01) $\text{Div}_d\ \text{Skel}^{d-1}\sigma^m \leftarrow \text{Skel}^{d-1}\sigma^m$; % each vertex is labeled by its name
(02) **for** $k$ **from** $d$ **to** $m$ **do** % Construct $\text{Div}_d\ \text{Skel}^k\sigma^m$ %
(03)     **for each simplex** $\sigma^k$ in $\sigma^m$ **do**
(04)         insert a vertex $b$ in the barycenter of $\sigma^k$;
             % this barycenter is labeled with the set of vertices of $\sigma_k$
(05)         construct the cone with apex at $b$ over $\text{Div}_d\ \partial\sigma^k$;
             % over the already subdivided boundary of $\sigma^k$ %
(06)         add the cone to $\text{Div}_d\ \text{Skel}^k\sigma^m$
(07)     **end for loop**
(08) **end for loop**.
```

Fig. 2. Constructing the subdivision $\text{Div}_d\ \sigma^m$ of a simplex σ^m for the d-solo model

The next lemma follows from the fact that the construction of Div_d in Figure 2 corresponds exactly to the description given in Lemma 1, and the fact in the system there are n processes, so they can start with at most n different input values (so only the input simplexes in $\mathcal{I}^*$ of dimension at most $n-1$ are relevant).

Lemma 3. *If $\mathcal{P}_R^*$ is the R-round algorithm complex of a colorless algorithm in the d-solo model with input complex $\mathcal{I}^*$, then $\mathcal{P}_R^*$ is an R-iterated, d-dimensional subdivision of the $n-1$ skeleton of $\mathcal{I}^*$.*

Returning to the (d, R)-subdivision task, we can now justify its name, simply by recalling that its output complex is equal to the algorithm complex:

Lemma 4. *The (d, R)-subdivision task over $\mathcal{I}^*$ for n processes is a triple $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, where $\mathcal{O}^*$ is the R-iterated, d-dimensional subdivision of the $n-1$ skeleton of $\mathcal{I}^*$, and Δ^* is equal to the corresponding subdivision carrier map.*

4 What Can Be Computed in the Presence of Solo Executions?

This section presents a characterization of the colorless tasks that can be solved in each one of the d-solo models. Consider an r round colorless algorithm that solves the colorless task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$. At the end of the r-th round, processes have to decide an output value, by executing $\mathsf{dec\_val}(\ell s_i)$ in line 3. The result of $\mathsf{dec\_val}(\ell s_i)$ is a vertex in $\mathcal{O}^*$, and different processes may decide different vertices as long as they belong to the same simplex of $\mathcal{O}^*$. This means that $\mathsf{dec\_val}$ is a simplicial map from $\mathcal{P}_r^*$ to $\mathcal{O}^*$. Moreover, $\mathsf{dec\_val}$ is carried by Δ^*, in the sense that for $\sigma \in \mathcal{I}^*$: $\mathsf{dec\_val}(\mathcal{P}_r^*(\sigma)) \subseteq \Delta^*(\sigma)$, which means that for any input simplex σ, any r round execution ends in a simplex τ of $\mathcal{P}_r^*$, and the decision that the processes make in τ, form an output simplex $\mathsf{dec\_val}(\tau)$ of $\mathcal{O}^*$. This output simplex $\mathsf{dec\_val}(\tau)$ must be in $\Delta^*(\sigma)$, to satisfy the task's specification.

Theorem 1. *The colorless task $\mathcal{T}^* = (\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$ is solvable with n processes in the d-solo model by a colorless algorithm if and only if there is an $R \geq 0$ and a simplicial map $\phi : Div_d^R\ Skel^{n-1}\mathcal{I}^* \to \mathcal{O}^*$ carried byΔ^*.*

5 (d, ϵ)-Solo Approx. Agreement and Strict Hierarchy of Models

We now study the properties of the (d, R)-agreement task of Section 3.3 in terms of a precision parameter ϵ, showing that this task can be solved in the d-solo model while it cannot be solved in the $(d + 1)$-solo model.

Let ϵ be a positive real. The (d, ϵ)-solo approximate agreement problem (in short (d, ϵ)-SAA) is a generalization of the ϵ-*approximate agreement* problem [6]. The $(1, \epsilon)$-solo approximate agreement instance implies 2ϵ-approximate agreement. Assuming the input of each process is a point of the d-dimensional Euclidean space $\mathbb{R}^d$, (d, ϵ)-solo approximate agreement is defined by the following properties. (This definition is discussed and compared to other definitions in [14].)

- Validity. Any output lies within the convex hull of the inputs.
- Agreement. There is a set of processes S, $1 \leq |S| \leq d$, such that any process p_i that is not in S decides a value o_i (point) such that the Euclidean distance between o_i and CH is at most ϵ, where CH is the convex hull of the points decided by the processes in S.
- Termination. If a process p_i does not crash, it decides a value.

It follows from this definition that up to d processes are allowed to decide any set of points within the convex hull (as an example each of them may decide the point it proposes). These processes define the set S, and intuitively, the values they decide are collectively "represented" by their convex hull CH. Finally, the values decided by the other processes are constrained by the values decided by the processes in S.

The next theorem shows that, from a task solvability point of view, the d-solo model is stronger than the $(d + 1)$-solo model.

Theorem 2. *If the domain of the possible input values (a) is bounded and (b) contains a regular simplex of dimension d whose edge length is strictly greater than $2\epsilon d\sqrt{\frac{2d}{d+1}}$, then the (d, ϵ)-solo approximate agreement problem is solvable in $\mathcal{ACS}^d_{n,n-1}$but not in $\mathcal{ACS}^{d+1}_{n,n-1}$.*

6 Conclusion

A process executes solo when its computes its local result without knowing the input values of the other participating processes. This paper addressed round-based asynchronous wait-free executions in which up to d processes may execute solo in each round. Among several contributions, the paper presented a strict hierarchy of wait-free iterated models, called d-solo models, and a topology-based characterization of the colorless tasks which can be solved in such d-solo models, $1 \leq d \leq n$. The paper also introduced a colorless task, denoted (d, ϵ)-solo approximate agreement (a generalization of the classic approximate agreement task), which can be solved in the d-solo model and cannot be solved in the $(d + 1)$-solo model.

Acknowledgments. This work has been partially supported by the French ANR project DISPLEXITY devoted to computability and complexity in distributed computing), and a UNAM-PAPIIT grant.

References

1. Attiya, H., Bar-Noy, A., Dolev, D.: Sharing memory robustly in message passing systems. Journal of the ACM 42(1), 121–132 (1995)
2. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., Reischuk, R.: Renaming in an asynchronous environment. Journal of the ACM 37(3), 524–548 (1990)
3. Borowsky, E., Gafni, E.: Immediate atomic snapshots and fast renaming. In: Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC 1993), pp. 41–51 (1993)
4. Borowsky, E., Gafni, E.: A simple algorithmically reasoned characterization of wait-free computations. In: Proc. 16th ACM PODC 1997, pp. 189–198 (1997)
5. Chaudhuri, S.: More choices allow more faults: set consensus problems in totally asynchronous systems. Information and Computation 105, 132–158 (1993)
6. Dolev, D., Lynch, N.A., Pinter, S.S., Stark, E.W., Weihl, W.E.: Reaching approximate agreement in the presence of faults. Journal of the ACM 33(3), 499–516 (1986)
7. Elrad, T.E., Francez, N.: Decomposition of distributed programs into communication-closed layers. Science of Computer Programming 2(3), 155–173 (1982)
8. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM 32(2), 374–382 (1985)
9. Gafni, E., Guerraoui, R.: Generalized universality. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 17–27. Springer, Heidelberg (2011)
10. Gafni, E., Rajsbaum, S.: Distributed programming with tasks. In: Lu, C., Masuzawa, T., Mosbah, M. (eds.) OPODIS 2010. LNCS, vol. 6490, pp. 205–218. Springer, Heidelberg (2010)
11. Herlihy, M.P.: Wait-free synchronization. ACM TOPLAS 13(1), 124–149 (1991)
12. Herlihy, M.P., Rajsbaum, S.: The topology of shared-memory adversaries. In: Proc. 29th ACM Symp. on Principles of Distr. Computing (PODC 2010), pp. 105–113. ACM Press (2010)
13. Herlihy, M.P., Rajsbaum, S.: A classification of wait-free loop agreement tasks. Theoretical Computer Science 291(1), 55–77 (2003)
14. Herlihy, M.P., Rajsbaum, S., Raynal, M., Stainer, J.: Computing in the Presence of Concurrent Solo Executions. Tech. Report 2004, IRISA (France), 19 pages (2013)
15. Herlihy, M.P., Shavit, N.: The topological structure of asynchronous computability. Journal of the ACM 46(6), 858–923 (1999)
16. Loui, M.C., Abu-Amara, H.H.: Memory requirements for agreement among unreliable asynchronous processes. Adv. in Comp. Research, vol. 4, pp. 163–183. JAI Press (1987)
17. Mendes, H., Herlihy, M.P.: Multidimensional approximate agreement in Byzantine asynchronous systems. In: 45th ACM Symp. on the Theory of Comp. (STOC 2013), pp. 391–400 (2013)
18. Munkres, J.R.: Elements of algebraic topology, 547 pages. Addison-Wesley (1984)
19. Raynal, M.: Concurrent programming: algorithms, principles, and foundations, 515 pages. Springer (2013) ISBN 978-3-642-32027-9
20. Vaidya, N.H., Garg, V.K.: Byzantine vector consensus in complete graphs. In: Proc. 32nd ACM Symposium on Principles of Distributed Computing (PODC 2013). ACM Press (2013)
21. Zeeman, E.C.: Relative simplicial approximation. Proc. Mathematical Proceedings of the Cambridge Philosophical Society 60, 39–43 (1964)