A Characterization of t-Resilient Colorless Task Anonymous Solvability

Carole Delporte-Gallet[1], Hugues Fauconnier[1], Sergio Rajsbaum[2], and Nayuta Yanagisawa[3]

[1] IRIF-GANG-Université Paris-Diderot, France.
{cd,hf}@irif.fr *
[2] Instituto de Matemáticas, UNAM, Mexico.
rajsbaum@math.unam.mx **
[3] Dept. of Mathematics, Graduate School of Science, Kyoto University, Kyoto.
nayuta87@math.kyoto-u.ac.jp

Abstract. One of the central questions in distributed computability is characterizing the tasks that are solvable in a given system model. In the *anonymous* case, where processes have no identifiers and communicate through multi-writer/multi-reader registers, there is a recent topological characterization (Yanagisawa 2017) of the *colorless* tasks that are solvable when any number of asynchronous processes may crash.
In this paper, we consider the case where at most t asynchronous processes may crash, where $1 \leq t < n$. We prove that a colorless task is t-resilient solvable anonymously if and only if it is t-resilient solvable non-anonymously. We obtain our results through various reductions and simulations that explore how to extend techniques for non-anonymous computation to anonymous one.

Keywords: MWMR registers, Anonymity, Distributed task, Topology

1 Introduction

One of the central questions in distributed computability is characterizing the tasks which are solvable in a given system model. A *task* is the distributed equivalent of a function in sequential computing: each process starts with a private input value, communicates with other processes, and eventually decides an output value, such that the vector of output values is valid for the vector of input values according to the task specification.

The *asynchronous computability theorem* (ACT) [26] is one of the central results in distributed computability. It characterizes the tasks that are solvable in

* Supported by LiDiCo.
** Supported by UNAM-PAPIIT IN109917. Part of this work was done while visiting Université Paris-Diderot.

shared-memory systems where n processes that may fail by crashing communicate by reading and writing shared registers. It is sometimes called the *wait-free* characterization, because any number of processes may crash and the processes are asynchronous (run at arbitrary speeds, independent from each other). The characterization is of an algebraic topological nature. In terms of algebraic topology, a *task* is represented as a relation Δ between an input complex $\mathcal{I}$ and an output complex $\mathcal{O}$. Each simplex σ in $\mathcal{I}$ is a set that specifies the initial inputs to the processes in some execution. The processes communicate with each other, and eventually decide output values that form a simplex τ in $\mathcal{O}$. The computation is correct if τ is in $\Delta(\sigma)$. The complex $\mathcal{I}$ (resp. $\mathcal{O}$) is *chromatic* because each simplex specifies not only input values, but also which process gets which input (resp. output) value. Roughly, the ACT characterization states that the task is solvable if and only if there is a chromatic simplicial map δ from a chromatic subdivision of $\mathcal{I}$ to $\mathcal{O}$ respecting Δ.

The ACT is the basis to obtain a characterization of distributed task computability in the case where at most t asynchronous processes may crash, where $1 \leq t < n$. It is also the basis to study other failure, timing, and communication models, and even mobile robot models [30]. There are basically two ways of extending the results from the wait-free model to other models. One is by directly generalizing the algorithmic and topological techniques, and the other is by reduction to other models using simulations (either algorithmic [6] or topological [24]). An overview of results in this area can be found in the book [20].

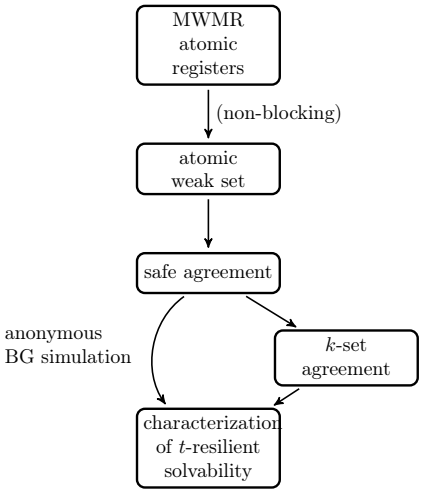
The theory of distributed computing presented in [20] assumes that the processes, $p_0, \ldots, p_{n-1}$, communicate using single-writer/multi-reader (SWMR) registers, $R_0, \ldots, R_{N-1}$. Thus, p_i knows that it is the i-th process and it can write exclusively to R_i while the size of the namespace, N, is assumed to be much bigger than the number of the process, n. In this situation, preallocating a register for each identifier would lead to a distributed algorithm with a very large space complexity, namely N registers. Instead, it is shown in [13] that n multi-writer/multi-reader (MWMR) registers are sufficient to solve any read-write wait-free solvable task.

However, in some distributed systems, processes are *anonymous*; they have no ids at all or they cannot make use of their identifiers (e.g., due to privacy issues). In such a system, processes run identical programs, and the means by which processes access the shared memory are identical to all processes. A process cannot have a private register to which only this process may write, and hence the shared memory consists only of MWMR registers. This anonymous shared memory model of asynchronous distributed computing has been studied since early on [3, 29], in the case where processes do not fail.

In an anonymous system, *colorless* tasks are natural, because they are defined only in terms of input and output values without stating which process receives which input value or which process produces which output value. Furthermore, the class of colorless tasks includes various important tasks, such as consensus and set agreement, and is rich enough to be undecidable even for three processes [17, 21].

Colorless tasks have been well studied in shared-memory and message passing models, where each process has a distinct identifier [20], but less so in anonymous systems. Only recently, the ACT has been extended to the anonymous case [31], providing a characterization of wait-free anonymous computability of colorless tasks. The characterization implies that the anonymity does not reduce the computational power of the asynchronous shared-memory model as far as colorless tasks are concerned. In consequence, the topological characterization is in terms of input and output complexes which are not chromatic.

Results Our main result is an extension of the wait-free characterization of [31] to the case where at most t processes may crash, where $1 \leq t < n$. We prove that a colorless task is t-resilient solvable anonymously if and only if it is t-resilient solvable non-anonymously. This implies a complete characterization of t-resilient, asynchronous, and anonymous computability of colorless tasks.



The result is obtained through a series of reductions depicted in the figure below. First, we design an anonymous non-blocking implementation of an atomic *weak set object* with n registers. The construction is based on the non-blocking atomic snapshot of [15, 19]. Then, we build a wait-free implementation of a safe agreement object for an arbitrary value set V. Our implementation is a generalization of the anonymous consensus algorithm proposed in [3]. We describe two ways of deriving the t-resilient anonymous solvability characterization. One way is through a novel anonymous implementation of the BG-simulation [6], which we use to simulate a non-anonymous system by an anonymous system, both t-resilient. The other way is to use the safe-agreement object to solve k-set agreement and then do the topological style of analysis [24, 31].

Related work Colorless tasks were first identified in [6]. They include fundamental tasks such as consensus [16], set agreement [11], and loop agreement [22], and have been widely studied in the non-anonymous setting. The first part of the book [20] is devoted to colorless tasks. Not all tasks of interest are colorless though, and general tasks can be much harder to study, notably renaming [9, 10].

A characterization of the colorless tasks that are solvable in the presence of processes that can crash in a dependent way is provided in [23], and a characterization when several processes can run solo is provided in [25]. Both encompass

the wait-free colorless task solvability characterization, and the former encompasses the t-resilient characterization that we use in this paper.

A certain kind of anonymity has been considered in [26] to establish the anonymous computability theorem. However, they allow the use of SWMR registers while we assume a fully anonymous model with only MWMR registers.

Anonymous distributed computing remains an active research area since the shared-memory seminal papers [3, 29] and the message-passing paper [1]. For some recent papers and references therein see, e.g. [8, 18].

Closer to our paper is [19] where the anonymous asynchronous MWMR fault-tolerant shared-memory model is considered. Our weak set object uses n MWMR registers and is non-blocking; it provides an enhanced atomic implementation of the weak set object supporting non-atomic operations presented in [12]. A wait-free implementation of a weak set object using $2n$ registers is in [14]. A set object that also supports a remove operation, but satisfies a weaker consistency condition, called per-element sequential consistency is presented in [4, 5].

Organization In Section 2 we briefly recall some of the notions used in this paper, about the model of computation and the topology tools, both of which are standard. In Section 3 we present the anonymous implementation of an atomic weak set object from MWMR registers. In Section 4 we present the safe agreement implementation. In Section 5 we derive our anonymous characterization of the t-resilient solvability of colorless tasks. Some proofs are omitted from this extended abstract for lack of space.

2 Preliminaries

We recall here briefly some standard notions of concurrent programming, for more precise definitions see [27]. We assume a standard *anonymous asynchronous shared-memory model* e.g., [19] consisting of n sequential processes that have no identifiers and execute an identical code. We assume that at most t of the processes may fail by crashing, where $1 \leq t < n$. Processes are asynchronous, i.e., they run at arbitrary speeds, independent from each other. We consider *linearizable* implementations where each operation appears to take effect instantaneously at some point between its invocation and response [28]. A *non-blocking* algorithm guarantees system-wide progress, while a *wait-free* also guarantees per-process progress. The processes communicate via multi-writer/multi-reader (MWMR) registers. Let $R[0 \ldots m-1]$ denote an array of m registers. The read operation, denoted by $\text{READ}(i)$, returns the state of $R[i]$. The write operation, denoted by $\text{WRITE}(i, v)$, changes the state of $R[i]$ to v and returns ack. The registers are assumed to be atomic (linearizable). We sometimes refer to the processes by unique names p_0, ..., p_{n-1} for the convenience of exposition, but processes themselves have no means to access these names. Let $\Pi = \{p_0, \ldots, p_{n-1}\}$.

3 Atomic Weak Set

Here, we present an anonymous implementation of an atomic weak set object on an arbitrary value set V.

3.1 Specification and Algorithm

An *atomic weak set* object, denoted by SET, is an atomic object used for storing a set of values. The object supports only two operations, ADD() and GET(), and has no remove operation, which is why it is called "weak." The ADD(v) operation takes an argument $v \in V$ and returns ACK. The GET() operation, takes no argument and returns the set of values that have appeared as arguments in all the ADD() operations preceding the GET() operation. We assume that SET initially holds no values, i.e., it holds $\emptyset$.

We assume that a non-blocking n-component atomic *snapshot* object is available. An implementation in an anonymous setting with n registers is described in [15, 19]. The snapshot object exports two operations, UPDATE() and SCAN(). Informally, an UPDATE(i, v) updates the i-th component of the object with the value v and a SCAN() returns an array of n values, which are contained in the n components at some point in time between the invocation and the response of the SCAN() operation.

We present an anonymous non-blocking implementation of the atomic weak set object on an n-component atomic snapshot object. The pseudocode of the implementation appears in Fig. 1. If $Snap$ is an array of n cells, we define $vals(Snap) = \cup_{i \in \{0,\ldots,n-1\}} Snap[i]$. The idea of the algorithm is as follows. To execute an ADD(v) operation, the algorithm repeatedly tries to store the value v in each one of the n components of the snapshot object, using an *update* operation (line 5) until it detects that v appears in all the components. In each iteration, the algorithm deposits in the snapshot object not only v but $View$ containing all the values known to be in the set so far. Once v is detected to be in all components of the snapshot object, the ADD(v) terminates. The GET() operation is similar, except that now the $View$ of the process has to appear in all the components of the snapshot for the operation to terminate. Intuitively, once a value v (or a set of values) appears in all n components of the snapshot object, it cannot be overwritten and cannot go unnoticed by other processes. This is because the other processes can be covering (about to overwrite) at most $n - 1$ components.

Theorem 1. *The algorithm of Fig. 1 is an anonymous non-blocking implementation of an atomic weak set object using n MWMR registers.*

Here is a sketch of the proof.

Given an operation op, $invoc(op)$ denotes its invocation and $resp(op)$ its response. Let H be a history of the algorithm as defined in [28]. Let H' be the history H in which some of the operations that are invoked by a process that crashes during the operation and doesn't get a response are removed. H_{seq} denotes the sequential history in which each operation of H' appears as if it has been executed at a single point (the linearization) of the time line.

Shared variable :
  n-component atomic snapshot object: R

Code for a process
Local variable:
  array of value sets $Snap[0 \dots n-1]$
  set of Values $View$ init $\emptyset$
  integer $next$

Macro:
  $vals(Snap) = \cup_{i=0}^{n-1} Snap[i]$

ADD(v):
1 $next = 0$
2 $Snap = R$.SCAN()
3 $View = View \cup vals(Snap) \cup \{v\}$
4 **while** $(\#\{r | v \text{ in } Snap[r]\} < n)$
5 R.UPDATE($next, View$)
6 $next = (next + 1) \bmod n$
7 $Snap = R$.SCAN()
8 $View = vals(Snap) \cup View$
9 **return** ACK

GET:
10 $next = 0$
11 $Snap = R$.SCAN()
12 $View = vals(Snap) \cup View$
13 **while** $(\#\{r | View = Snap[r]\} < n)$
14 R.UPDATE($next, View$)
15 $next = (next + 1) \bmod n$
16 $Snap = R$.SCAN()
17 $View = vals(Snap) \cup View$
18 **return** $View$

Fig. 1. Non-blocking implementation of atomic weak set for n processes.

Safety For the safety part, we have to define linearization points and prove that

- the linearization point of each operation GET() and ADD() appear between the beginning and the end of this operation;
- the sequential history that we get with these points respects the sequential specification of the weak set.

Consider a history H, let v be a value or a set of values. Define time τ_v as the first time, if any, that v belongs to all components of R. When there is no such time, τ_v is $\perp$.

Lemma 1. *If the operation* ADD(v) *terminates, then v belongs to all components of R at some time instance before the end of this operation. If the operation* GET() *terminates and returns V, then V belongs to all components of R before the end of this operation.*

By Lemma 1, τ_v is not $\perp$ for each operation ADD(v) that terminates and τ_V is also not $\perp$ for each operation GET() that terminates and returns V.

It can be shown that the linearization points for operations ADD() and GET() are as follows

- op = ADD(v): If $\tau_v \neq \perp$, the linearization point τ_{op} of an operation op = ADD(v) is $max\{\tau_v, invoc(op)\}$. If $\tau_v = \perp$, the operation op does not terminate and is not linearized.
- op = GET(): The linearization point τ_{op} of an operation op = GET() that returns V is $max\{\tau_V, invoc(op)\}$. A GET() operation that does not terminate is not linearized.

The main safety claim is the following.

Lemma 2. *H_{seq} satisfies the sequential specification of the weak set.*

Liveness We prove that the algorithm is non-blocking; namely, if processes perform operations forever, an infinite number of operations terminate. By contradiction, assume that there is only a finite number of operations GET() and ADD() that terminate and some operations made by correct processes do not terminate.

Operations ADD() or GET() may not terminate because the termination conditions of the while loop are not satisfied (Lines 4 or 13): for an ADD(v) operation, in each SCAN() made by the process, v is not in at least one of the components of R, and for a GET() operation, in each *snap*, all the components are not equal to the view of the process.

There is a time τ_0 after which there is no new process crash and all processes that terminate GET() or ADD() operations in the run have already terminated. Consider the set N of processes alive after time τ_0 that do not terminate operations in the run. Notice that after time τ_0 only processes in N take steps. Also, as no process in N may crash, each process in N takes an infinite number of steps.

Notice that all values in variables $View$ have been proposed by some ADD(). If there is a finite number of operations, then all variables $View$ are subsets of a finite set of values. The main idea of the liveness proof is to analyze *stable views*. That is, the sequence of views of each process is non-decreasing, each two consecutive views satisfy $view \subseteq view'$. Thus, there is a time $\tau_1 > \tau_0$ after which the view of each process p in N converges to a *stable view* $SView_p$: forever after time τ_1 the view of p is $SView_p$. Let $SV = \{SView_p | p \in N\}$, be the set of all stable views for processes in N. It can be shown that there is no minimal stable view, proving that $SV = \emptyset$ and also $N = \emptyset$, a contradiction.

4 Safe Agreement Object

A *safe agreement object* [6] on a set V provides two operations, *propose* and *resolve*. A propose operation, denoted by PROPOSE(v), takes an argument $v \in V$ and returns ACK. A resolve operation, denoted by RESOLVE(), takes no argument and returns $u \in V$ or $\perp$, where $\perp \notin V$. An execution is *well-formed* if each

process invokes at most one propose operation and no process invokes a propose or resolve operation before its previous operation has terminated. In any well-formed execution, the object satisfies the following four conditions [2, 7]:

Validity Any non-$\perp$ value returned by a resolve operation is an argument of some propose operation;
Agreement If two resolve operations return non-$\perp$ values v and v', then $v = v'$;
Termination Every operation invoked by a non-faulty process eventually terminates;
Nontriviality If no process fails while performing its propose operations, every resolve operation started after some time instance returns a non-$\perp$ value.

An anonymous wait-free implementation of a safe agreement object for an arbitrary value set V is presented in Fig. 2. The implementation makes use of an array of n weak set objects, denoted by $SET[0 \dots n-1]$. To perform a propose operation, each process first assigns its input value to a local variable *view*. Then, the process repeats the following procedure for $i = 0, \dots, n-1$: it adds *view* to $SET[i]$; if $SET[i]$ holds a set of cardinality more than one and *view* is the minimum value of the set, it returns ACK and immediately breaks the loop; otherwise, it assigns the minimum value of the set to *view*. To perform a resolve operation, each process checks the set held by $SET[n-1]$. If the set is not empty, the process returns the minimum value in the set. Otherwise, the process returns $\perp$.

Our implementation is a generalization of the anonymous consensus algorithm proposed by Attiya et al. [3]. Bouzid and Corentin [7] have proposed an anonymous implementation of a safe agreement object for the case of $V = \{0, 1\}$, also based on [3]. However, their implementation does not (directly) extend to the case of an infinite value set.

We now sketch the correctness proof of the algorithm of Fig. 2. Recall that, although we refer to the processes by unique names p_0, ..., p_{n-1}, processes do not know these names and that $\Pi = \{p_0, \dots, p_{n-1}\}$.

Lemma 3. *Fix a well-formed execution of the algorithm of Fig. 2. Let V_i be the set of all the values that are added to $SET[i]$ in the execution. Then, $V_i \supseteq V_{i+1}$ holds for all $i = 0, \dots, n-2$.*

Lemma 4 (Validity). *The algorithm of Fig. 2 satisfies the validity condition.*

Lemma 5. *Fix a well-formed execution of the algorithm of Fig 2. Let V_i be the set of the all values that are added to $SET[i]$ in the execution. Let us define*

$$\Pi_i = \{p \in \Pi \mid p \text{ performs } \textsc{propose}() \text{ and adds some } v \in V_i \setminus \{\min V_i\} \text{ to } SET[i]\}.$$

Then, $\Pi_i \supseteq \Pi_{i+1}$ holds for all $i = 0, \dots, n-2$.

Lemma 6 (Agreement). *The algorithm of Fig. 2 satisfies the agreement condition.*

Shared variable :
`array of atomic weak set objects` : $SET[0 \dots n-1]$

CODE FOR A PROCESS

Local variable:
`Value` *view* init $\perp$
`Integer` i init 0
`set of Values` *Snap* init $\emptyset$

operation PROPOSE(v):
1 $view = v$
2 **for** $i = 0, \dots, n-1$ **do**
3 $SET[i]$.ADD($view$)
4 $Snap = SET[i]$.GET()
5 **if** $\#Snap \geq 2$ && $view = \min(Snap)$ **then**
6 **return** ACK
7 **else**
8 $view = \min(Snap)$
9 **return** ACK

operation RESOLVE():
10 $Snap = SET[n-1]$.GET()
11 **if** $Snap \neq \emptyset$ **then**
12 **return** $\min(Snap)$
13 **else**
14 **return** $\perp$

Fig. 2. Anonymous implementation of safe agreement object

Lemma 7 (Termination). *The algorithm of Fig. 2 satisfies the termination condition.*

Lemma 8 (Nontriviality). *The algorithm of Fig. 2 satisfies the nontriviality condition.*

By Lemmas 4, 6, 7, and 8, we obtain the following theorem. Furthermore, notice that the algorithm uses n atomic registers, because an arbitrary finite number of atomic weak set objects can be simulated on top of a single atomic weak set object.

Theorem 2. *The algorithm of Fig. 2 is an anonymous wait-free implementation of safe agreement object, using n atomic registers.*

5 t-Resilient Solvable Colorless Tasks

We give a characterization of t-resilient solvable colorless tasks (see formal definitions below) in the anonymous shared-memory model.

Theorem 3. *A colorless task is t-resilient solvable in the anonymous shared-memory model if and only if it is t-resilient solvable in the non-anonymous one. Moreover, if a colorless task is t-resilient solvable by n anonymous processes, it can be solved by n shared atomic registers.*

The only if part of the theorem is immediate because every anonymous protocol can be executed by non-anonymous processes. We next describe the if part by two different approaches, a topological one and an operational one.

5.1 Topological Approach

We briefly recall some notions of combinatorial topology for distributed computing, additional details can be found in [20].

Let I and O be complexes. A *carrier map* from $\mathcal{I}$ to $\mathcal{O}$ is a mapping $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$ such that, for each $s \in \mathcal{I}$, $\Delta(s)$ is a subcomplex of $\mathcal{O}$ and $s' \subseteq s$ implies $\Delta(s') \subseteq \Delta(s)$. If a continuous map $f : |\mathcal{I}| \to |\mathcal{O}|$ satisfies $f(|\sigma|) \subseteq |\Delta(\sigma)|$ for all $\sigma \in \mathcal{I}$, we say that f is carried by Δ. If a simplicial map $\delta : \text{bary}^b\mathcal{I} \to \mathcal{O}$ satisfies $\delta(\text{bary}^b\sigma) \subseteq \Delta(\sigma)$ for all $\sigma \in \mathcal{I}$, we say that δ is *carried by* Δ. As an immediate consequence of Lemma 3.7.8. of [20], the following lemma holds.

Lemma 9. *If $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$ is a carrier map and $f : |\mathcal{I}| \to |\mathcal{O}|$ is a continuous map carried by Δ, then there is a non-negative integer b and a simplicial map $\delta : bary^b\mathcal{I} \to \mathcal{O}$ carried by Δ.*

A *colorless task* is a triple $T = (\mathcal{I}, \mathcal{O}, \Delta)$, where $\mathcal{I}$ and $\mathcal{O}$ are simplicial complexes and Δ is a carrier map. A colorless task T is solvable, if for each input simplex $s \in \mathcal{I}$, whenever each process p_i starts with input value $v_i \in s$ (different processes may start with the same value), eventually it decides an output value v_i', such that the set of output values form a simplex $s' \in \Delta(s)$. The colorless tasks that are fundamental to the present paper are *b-iterated barycentric agreement* and *k-set agreement*. The b-iterated barycentric agreement task is a colorless task $T = (\mathcal{I}, \text{bary}^b\mathcal{I}, \text{bary}^b)$, where we write by bary^b the carrier map that maps $s \in \mathcal{I}$ to $\text{bary}^b s$ for an abuse of notation. The k-set agreement task is a colorless task $T_k = (\mathcal{I}, \text{skel}^k\mathcal{I}, \text{skel}^k)$, where skel^k denotes the carrier map that maps a simplex $s \in \mathcal{I}$ to the subcomplex $\text{skel}^k\mathcal{I}$.

To prove the if part of Theorem 3, we first show that the $(t+1)$-set agreement task is t-resilient solvable by n anonymous processes. An algorithm of Fig. 3 presents an anonymous t-resilient protocol for the $(t+1)$-set agreement task. In the protocol, each process first proposes its input value to $SA[i]$ for $i = 0, \ldots, t$. Then, the process repeatedly performs a RESOLVE() operation to all $SA[i]$ in the round-robin manner until it gets non-$\perp$ value. Once the process gets non-$\perp$ value, the process returns the value.

Theorem 4. *The algorithm of Fig. 3 is a t-resilient anonymous protocol for the $(t+1)$-set agreement task.*

Proof. Termination: It suffices to show that the while loop of Line 4–6 eventually terminates. In the protocol, each process performs PROPOSE() operations to

Shared variable :
`array of safe agreement objects` : $SA[0 \dots t]$

CODE FOR A PROCESS

Local variable:
`Integer` i init 0
`Value` $result$ init $\perp$

SETAGREE(v):
1 **for** $i = 0, \dots, t$ **do**
2 $SA[i]$.PROPOSE(v)
3 $i = 0$
4 **while** $result = \perp$ **do**
5 $result = SA[i]$.RESOLVE()
6 $i = i + 1 \mod t + 1$
7 **return** $result$

Fig. 3. Anonymous t-resilient $(t+1)$-set agreement protocol

$SA[0], \dots, SA[t]$ sequentially. Thus, even if t processes fail, there is at least one safe agreement object such that no process fails while performing a PROPOSE() operation on the object. By the nontriviality property of safe agreement objects, after some time instance, RESOLVE() operations on some safe agreement object return non-$\perp$ value and thus the while loop eventually terminates.

Validity: Every argument of a PROPOSE() operation is a proposed value. Because of the validity property of safe agreement objects, a non-$\perp$ value returned by some RESOLVE() operation is one of the arguments of PROPOSE() operations. Thus, the validity condition holds.

k-Agreement: There are $t+1$ distinct safe agreement objects. Thus, by the agreement property of safe agreement objects, at most $t+1$ distinct values are decided.

As the b-iterated barycentric agreement task is wait-free solvable by anonymous processes [31], the following lemma holds.

Lemma 10. *Let $T = (\mathcal{I}, \mathcal{O}, \Delta)$ be a colorless task. If there exists a continuous map $f : |skel^t \mathcal{I}| \to |\mathcal{O}|$ carried by Δ, T is t-resilient solvable by n anonymous processes.*

Proof. By Lemma 9, there is an integer b and a simplicial map $\delta : \text{bary}^b \text{skel}^t \mathcal{I} \to \mathcal{O}$ that satisfies $\delta(\text{bary}^b \sigma) \subseteq \Delta(\sigma)$ for every $\sigma \in \text{skel}^t \mathcal{I}$.

The following anonymous protocol solves the colorless task. Suppose that the set of all inputs to the processes is $s \in \mathcal{I}$. Execute first the anonymous $(t+1)$-set agreement protocol so that the processes all choose vertices that form a simplex sigma in $\text{skel}^t \mathcal{I}$, and then the b-iterated barycentric agreement protocol (for a sufficiently large value of b). Each process chooses a vertex of a common simplex of $\text{bary}^b \text{skel}^t \sigma$. Finally, each process determines its output by applying δ to the vertex it chose.

Shared variable :
`atomic weak set` : SET
`array of safe agreement objects` : $SA[0\ldots][0\ldots n-1]$

CODE FOR A PROCESS

Local variable:
`Value` $view_i$ init $\perp$ for $i = 0, \ldots, n-1$
`Integer` $round_i$ init 0 for $i = 0, \ldots, n-1$
`Integer` i init 0
`Value` $snap$ init $\perp$

Simulation(v):
1 **for** $i = 0, \ldots, n-1$ **do**
2 $SA[0][i]$.PROPOSE(v)
3 **while** true **do**
4 **for** $i = 0, \ldots, n-1$ **do**
5 $view_i = SA[round_i][i]$.RESOLVE()
6 **if** $view_i$ is a termination state of P_i **then**
7 **return** $f(view_i)$
8 **elseif** $view_i \neq \perp$ **then**
9 SET.ADD($(P_i, view_i, round_i)$)
10 $snap = SET$.GET()
11 $view_i = latest\_views(snap)$
12 $round_i = latest\_round_i(snap) + 1$
13 $SA[round_i][i]$.PROPOSE($view_i$)

Fig. 4. n anonymous processes simulates n non-anonymous processes

The if part of Theorem 3 follows from Lemma 10 and the following theorem by Herlihy and Rajsbaum:

Theorem 5 ([23, Theorem 4.3]). *A colorless task $T = (I, O, \Delta)$ is t-resilient solvable by n non-anonymous processes if and only if there exists a continuous map $f : |skel^t I| \to |O|$ carried by Δ.*

Note that the protocol in the proof of Lemma 10 only makes use of a finite number of atomic weak set objects, which are constructed on top of a single atomic weak set object. Thus, every colorless task that is t-resilient solvable by n anonymous processes is solved with n atomic registers. The space complexity upper bound of Theorem 3 follows.

5.2 Simulation-Based Approach

We now prove the if part of Theorem 3 by a simulation, which is an anonymous variant of the BG-simulation [6]. More precisely, we show that n anonymous t-resilient processes with atomic weak set objects can simulate n non-anonymous t-resilient processes with *atomic snapshot objects*. We denote the anonymous simulators by p_0, ..., p_{n-1} and the non-anonymous simulated processes by P_0,

$\ldots, P_{n-1}$. Without loss of generality, we may assume that non-anonymous processes communicate via a single n-ary atomic snapshot object and execute a *full-information protocol*. In the protocol, the process P_i repeatedly writes its local state to the i-th component of the array, takes a snapshot of the whole array and updates its state by the result of the snapshot until it reaches a termination state. When the process reaches the termination state, it decides on the value obtained by applying some predefined function f to the state.

Our simulation algorithm for each simulator is presented in Fig. 4. The algorithm makes use of a two dimensional array of safe agreement objects $SA[0\ldots][0\ldots n-1]$, where the column $SA[0\ldots][i]$ is for storing simulated states of the process P_i. The local variables $view_i$ and $round_i$ stand for the current simulated state and the current simulated round of P_i respectively. The function $latest\_views$ maps a set of tuples consisting of a process name, its simulated state, and its simulated round to the array whose i-th component is the simulated view of P_i associated with the largest simulated round number of P_i. The function $latest\_round_i$ maps a set of the same kind to the latest round number of P_i.

In the algorithm, each simulator first proposes its input value to $SA[0][i]$ for all $P_0, \ldots, P_{n-1}$. Then, a simulator repeats the following procedure for $P_0, \ldots, P_{n-1}$ in the round-robin manner until one of $P_0, \ldots, P_{n-1}$ reach a termination state: it performs RESOLVE() operation on $SA[round_i][i]$; if the return value of the RESOLVE() operation is not $\perp$, the simulator adds the return value, with the name P_i and its current simulated round, to SET, updates simulated state and round, and proposes the new simulated state of P_i to $SA[round_i][i]$.

By the use of safe agreement objects, simulators can agree on the return value of each simulated snapshot. Note that there is no need to use a safe agreement object on each simulated update because each value to be updated is deterministically determined by the return value of the preceding simulated snapshot. In the algorithm of Fig. 4, each simulator performs PROPOSE() operations sequentially. Thus, even though t simulators crash, they block at most t simulated processes by the nontriviality property of the safe agreement object. By these observations, we establish the following lemma:

Lemma 11. *If a colorless task is t-resilient solvable by n non-anonymous processes with atomic snapshot objects, it is also t-resilient solvable by n anonymous processes with atomic weak set objects.*

The proof of the lemma is similar to the proof of Theorem 5 in [6], while we omit the proof.

The space complexity of the simulation of Fig. 4 is exactly n atomic registers because a single atomic weak set object can simulate, in the non-blocking manner, an arbitrary finite number of atomic weak set objects and safe agreement objects. This establishes the space complexity upper bound of Theorem 3.

6 Conclusion

In this paper, we have extended the wait-free colorless task solvability of [31] to the case where at most t processes may crash, where $1 \leq t < n$. Furthermore, we

have shown that any t-resilient solvable colorless task can be t-resilient solvable anonymously using only n MWMR registers. We have derived our result through a series of reductions that seem interesting in themselves, to study anonymous computability. We hope they are useful to study further long-lived objects (as opposed to tasks), perhaps using a wait-free implementation of the weak set object [14], and uniform solvability (instead of a fixed number of processes n). Also, it would be interesting to look for lower bounds on the number of MWMR registers needed to solve specific colorless tasks.

References

1. Dana Angluin. Local and global properties in networks of processors. In *12th Annual ACM Symposium on Theory of Computing (STOC)*, pages 82–93, 1980.
2. Hagit Attiya. Adapting to Point Contention with Long-Lived Safe Agreement. In *13th Int. Conf. Structural Information and Communication Complexity (SIROCCO)*, volume 4056 of *LNCS*, pages 10–23, 2006.
3. Hagit Attiya, Alla Gorbach, and Shlomo Moran. Computing in totally anonymous asynchronous shared memory systems. *Information and Computation*, 173(2):162–183, 2002.
4. Roberto Baldoni, Silvia Bonomi, and Michel Raynal. Value-based sequential consistency for set objects in dynamic distributed systems. In *16th International Euro-Par Conference (Euro-Par)*, volume 6271 of *LNCS*, pages 523–534, 2010.
5. Roberto Baldoni, Silvia Bonomi, and Michel Raynal. Implementing set objects in dynamic distributed systems. *Journal of Computer and System Sciences*, 82(5):654 – 689, 2016.
6. Elizabeth Borowsky, Eli Gafni, Nancy Lynch, and Sergio Rajsbaum. The BG distributed simulation algorithm. *Distributed Computing*, 14(3):127–146, 2001.
7. Zohir Bouzid and Corentin Travers. Anonymity-Preserving Failure Detectors. In *30th International Symposium Distributed Computing (DISC)*, volume 9888 of *LNCS*, pages 173–186, 2016.
8. Claire Capdevielle, Colette Johnen, Petr Kuznetsov, and Alessia Milani. On the uncontended complexity of anonymous agreement. *Distributed Computing*, 30(6):459–468, 2017.
9. Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. The renaming problem in shared memory systems: An introduction. *Comput. Sci. Rev.*, 5(3):229–251, 2011.
10. Armando Castañeda, Damien Imbs, Sergio Rajsbaum, and Michel Raynal. Generalized symmetry breaking tasks and nondeterminism in concurrent objects. *SIAM Journal on Computing*, 45(2):379–414, 2016.
11. Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132 – 158, 1993.
12. Carole Delporte-Gallet and Hugues Fauconnier. Two consensus algorithms with atomic registers and failure detector Ω. In *10th International Conference on Distributed Computing and Networking (ICDCN)*, volume 5408 of *LNCS*, pages 251–262, 2009.
13. Carole Delporte-Gallet, Hugues Fauconnier, Eli Gafni, and Sergio Rajsbaum. Linear space bootstrap communication schemes. *Theoretical Computer Science*, 561(Part B):122 – 133, 2015. Special Issue on Distributed Computing and Networking.

14. Carole Delporte-Gallet, Hugues Fauconnier, Sergio Rajsbaum, and Nayuta Yanagisawa. An anonymous wait-free weak-set object implementation. In *6th International Conference on Networked Systems (NETYS)*, volume to appear of *LNCS*, 2018.
15. Faith Ellen, Panagiota Fatourou, and Eric Ruppert. The space complexity of unbounded timestamps. *Distributed Computing*, 21(2):103–115, 2008.
16. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
17. Eli Gafni and Elias Koutsoupias. Three-processor tasks are undecidable. *SIAM Journal on Computing*, 28(3):970–983, 1999.
18. Rati Gelashvili. On the Optimal Space Complexity of Consensus for Anonymous Processes. In *30th International Symposium Distributed Computing (DISC)*, LNCS, pages 452–466, 2015.
19. Rachid Guerraoui and Eric Ruppert. Anonymous and fault-tolerant shared-memory computing. *Distributed Computing*, 20(3):165–177, 2007.
20. Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed computing through combinatorial topology*. Morgan Kaufmann, 2013.
21. Maurice Herlihy and Sergio Rajsbaum. The decidability of distributed decision tasks (extended abstract). In *29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 589–598, 1997.
22. Maurice Herlihy and Sergio Rajsbaum. A classification of wait-free loop agreement tasks. *Theoretical Computer Science*, 291(1):55 – 77, 2003.
23. Maurice Herlihy and Sergio Rajsbaum. The topology of shared-memory adversaries. In *29th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 105–113, 2010.
24. Maurice Herlihy and Sergio Rajsbaum. Simulations and reductions for colorless tasks. In *31th ACM symposium on Principles of distributed computing*, PODC '12, pages 253–260, New York, NY, USA, 2012. ACM.
25. Maurice Herlihy, Sergio Rajsbaum, Michel Raynal, and Julien Stainer. From wait-free to arbitrary concurrent solo executions in colorless distributed computing. *Theor. Comput. Sci.*, 683:1–21, 2017.
26. Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, November 1999.
27. Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2008.
28. Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, July 1990.
29. Prasad Jayanti and Sam Toueg. Wakeup under read/write atomicity. In *4th International Workshop on Distributed Algorithms*, pages 277–288, 1991.
30. Sergio Rajsbaum, Armando Castañeda, David Flores-Peñaloza, and Manuel Alcantara. Fault-tolerant robot gathering problems on graphs with arbitrary appearing times. In *31th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 493–502, May 2017.
31. Nayuta Yanagisawa. Wait-free solvability of colorless tasks in anonymous shared-memory model. *Theory of Computing Systems*, pages 1–18, 2017.