Affine Tasks for k-Test-and-Set

Petr Kuznetsov, Thibault Rieutord

Affine Tasks for k-Test-and-Set

Petr Kuznetsov * Thibault Rieutord*

Abstract

The paper proposes a surprisingly simple characterization of a classical class of models of distributed computing, captured by an *affine task*: A subcomplex of the second iteration of the standard chromatic subdivision. We show that the class of affine task we propose has an element equivalent, regarding task solvability, to any wait-free shared-memory model in which processes have additionally access to *k-test-and-set* objects. Our results thus extend existing affine characterization beyond *fair* models.

Distributed computing deals with a jungle of models, parameterized by types of failures, synchrony assumptions and communication primitives. Determining relative computability power of these models ("is model A more powerful than model B?") is an intriguing and important problem.

This paper deals with *shared-memory* models in which a set of *crash-prone asynchronous* processes communicate via invoking operations on a collection of shared objects, which, by default, include atomic read-write registers.

Topology of wait-freedom. The *wait-free* model [15] makes no assumptions on when and where failures might occur. Herlihy and Shavit proposed an elegant characterization of wait-free (read-write) task computability via the existence of a specific *simplicial* map from geometrical structures describing inputs and outputs [18].

A task T has a wait-free solution using read-write registers if and only if there exists a simplicial, chromatic map from some *subdivision* of the *input simplicial complex*, describing the inputs of T, to the *output simplicial complex*, describing the outputs of T, respecting the task specification Δ. In particular, we can choose this subdivision to be a number of iterations of the *standard chromatic* subdivision (denoted Chr, Figure 1). Therefore, the celebrated *Asynchronous Computability Theorem (ACT)* [18] can be formulated as:

> A task $T = (\mathcal{I}, \mathcal{O}, \Delta)$, where $\mathcal{I}$ is the input complex, $\mathcal{O}$ is an output complex, and Δ is a carrier map from $\mathcal{I}$ to sub-complexes of $\mathcal{O}$, is wait-free solvable if and only if there exists a natural number ℓ and a simplicial map $\phi : \mathrm{Chr}^\ell(\mathcal{I}) \to \mathcal{O}$ carried by Δ (informally, respecting the task specification Δ).

The output complex of the *immediate snapshot* (IS) task is precisely captured by Chr [4]. By solving the *IS* task iteratively, where the current iteration output is used as the input value for the next one, we obtain the *iterated* immediate snapshot (IIS) model, captured by iterations of Chr. The ACT theorem can thus be interpreted as: the set of wait-free (read-write) solvable task is precisely the set of tasks solvable in the IIS model. The ability of (iteratively) solving the *IS* task thus allows us to solve any task in the wait-free model. Hence, from the task computability perspective, the *IS* task is a finite representation of the wait-free model.

*LTCI, Télécom ParisTech, Université Paris-Saclay

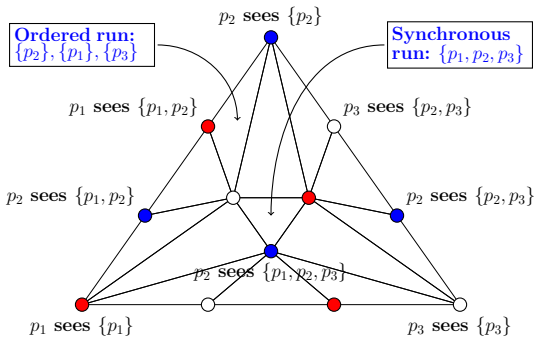

Figure 1: Chr(**s**), the standard chromatic subdivision of a 2-simplex, the output complex of the 3-process *IS* task.

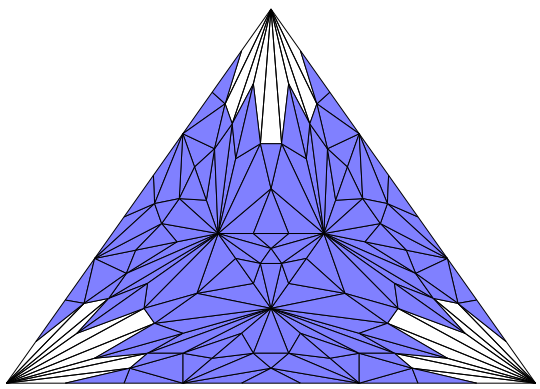

Figure 2: In blue, $\mathcal{R}_{1-res}$, the affine task of 1-resilience.

Adversaries. Given that many fundamental tasks are not solvable in the wait-free way [2, 18, 23], more general models were considered. Delporte et al. [8] introduced the notion of an *adversary*, a collection $\mathcal{A}$ of process subsets, called *live sets*. A run is in the corresponding *adversarial* $\mathcal{A}$*-model* if the set of processes taking infinitely many steps in it is a live set of $\mathcal{A}$. For example, the t*-resilient* n-process model is defined via an adversary $\mathcal{A}_{t-res}$ that consists of all process subsets of size $n-t$ or more. $\mathcal{A}_{t-res}$ is *superset-closed* [20], as it contains all supersets of its elements.

Saraph et al. [24] recently proposed a direct characterization of t-resilient task computability via a specific task $\mathcal{R}_{t-res}$. The task is defined as a restriction of the *double* immediate snapshot task: the output complex of the task is a sub-complex consisting of *all* simplices of the second degree of the standard chromatic subdivision of the task's input complex, except the simplices adjacent to the $(n-t-1)$-skeleton of the input complex. Intuitively the output complex of $\mathcal{R}_{t-res}$ contains all of 2-round *IS* runs in which every process "sees" at least $n-t-1$ other processes. We call such tasks *affine* [11, 13], as the geometrical representation of their output complexes are unions of affine spaces. An affine task consists in solving a (generalized) simplex agreement [4, 18] on the corresponding sub-complex of $\mathrm{Chr}^2\,\mathbf{s}$.

Figure 2 depicts the output complex of $\mathcal{R}_{1-res}$, the affine task for the 3-process 1-resilient model.

Solving a task T in the t-resilient model is then equivalent to finding a map from iterations of

$\mathcal{R}_{t-res}$ (applied to the input complex of T) to the output complex of T.

Similarly, the affine task of the *k-obstruction-free* adversary, consisting of all process subsets of size at most k, was recently determined by Gafni et al. [11]. Note that such an adversary is *symmetric* [26], as it only depends on the *sizes* of live sets, and not on process identifiers. Unlike $\mathcal{A}_{t-res}$ (which is also symmetric), the k-obstruction-free one is not superset-closed.

Topology of fair adversaries. In this paper, we present a compact topological characterization of wait-free shared memory models enhanced with k-test-and-set objects. We define an affine task $\mathcal{R}_{k-T\&S}$ capturing the task computability of any wait-free shared memory models enhanced with k-test-and-set objects. Our characterization can be put as the following generalization of the ACT [18]:

> A task $T = (\mathcal{I}, \mathcal{O}, \Delta)$ is solvable in a wait-free shared memory models enhanced with k-test-and-set objects. if and only if there exists a natural number ℓ and a simplicial map $\phi : \mathcal{R}^{\ell}_{k-T\&S}(\mathcal{I}) \to \mathcal{O}$ carried by Δ.

We believe that the results can be extended to all "practical" restrictions of the wait-free model which may result in a complete computability theory for distributed computing shared-memory models.

Roadmap. Section 1 describes our model. Section 2 defines the affine task $\mathcal{R}_{k-T\&S}$. In Section 3, we show that $\mathcal{R}^{*}_{k-T\&S}$ can be simulated in the wait-free shared memory model enhanced with k-test-and-set objects. In Section 4, we show that reciprocally any task solvable in the wait-free shared memory model enhanced with k-test-and-set objects can be solved in $\mathcal{R}^{*}_{k-T\&S}$. Section 7 reviews related work and Section 8 concludes the paper.

1 Preliminaries

We assume a system of n asynchronous processes, $\Pi = \{p_1, \ldots, p_n\}$. Two models of communication are considered: (1) *atomic snapshots* [1] and (2) *iterated immediate snapshots* [4, 18].

Atomic snapshot models. The atomic-snapshot (AS) memory is represented as a vector of n shared variables, where each process p_i is associated with the position i. The memory can be accessed with two operations: *update* and *snapshot*. An *update* operation performed by p_i modifies the value at position i and a *snapshot* returns the vector current state.

A *protocol* is a deterministic distributed automaton that, for each process and each its local state, stipulates which operation and state transition the process may perform. A *run* of a protocol is a possibly infinite sequence of alternating states and operations. An AS *model* is a set of infinite runs.

In an infinite run of the AS model, a process that takes only finitely many steps is called *faulty*, otherwise it is called *correct*. We assume that in its first step, a process shares its initial state using the *update* operation. If a process completed this first step in a given run, it is said to be *participating*, and the set of participating processes is called the *participating set*. Note that every correct process is participating.

(a) Ordered run: $\{p_2\}, \{p_1\}, \{p_3\}$.

(b) Synchronous run: $\{p_1, p_2, p_3\}$.

Figure 3: Examples of valid sets of IS outputs.

Iterated Immediate Snapshot model. In the iterated immediate snapshot (IIS) model, processes proceed through an infinite sequence of independent memories $M_1, M_2, \ldots$. Each memory M_r is accessed by a process with a single *WriteSnapshot* operation [3]: the operation performed by p_i takes a value v_{ir} and returns a set V_{ir} of submitted values (w.l.o.g, values of different processes are distinct), satisfying the following properties (See Figure 3 for IS examples):

- self-inclusion: $v_{ir} \in V_{ir}$;
- containment: $(V_{ir} \subseteq V_{jr}) \vee (V_{jr} \subseteq V_{ir})$;
- immediacy: $v_{ir} \in V_{jr} \Rightarrow V_{ir} \subseteq V_{jr}$.

In the IIS communication model, we assume that processes run the *full-information* protocol, in which, the first value each process writes is its *initial state*. For each $r > 1$, the outcome of the WriteSnapshot operation on memory M_{r-1} is submitted as the input value for the WriteSnapshot operation on M_r. There are no failures in the IIS model, all processes go through infinitely many *IS* instances.

Note that the wait-free AS model and the IIS model are equivalent as regards task solvability [3, 16].

Tasks. In this paper, we focus on distributed *tasks* [18]. A process invokes a task with an *input* value and the task returns an *output* value, so that the inputs and the outputs across the processes respect the task specification. Formally, a *task* is defined through a set $\mathcal{I}$ of input vectors (one input value for each process), a set $\mathcal{O}$ of output vectors (one output value for each process), and a total relation $\Delta : \mathcal{I} \mapsto 2^{\mathcal{O}}$ that associates each input vector with a set of possible output vectors. We require that Δ is a *carrier* map: $\forall \rho, \sigma \in \mathcal{I}$, $\rho \subseteq \sigma$: $\Delta(\rho) \subseteq \Delta(\sigma)$. An input $\perp$ denotes a *non-participating* process and an output value $\perp$ denotes an *undecided* process. Check [16] for more details on the definition.

In the *k-set consensus* task [6], input values are in a set of values V ($|V| \geq k+1$), output values are in V, and for each input vector I and output vector O, $(I, O) \in \Delta$ if the set of non-$\perp$ values in O is a subset of values in I of size at most k. The case of 1-set consensus is called *consensus* [9].

A protocol solves a task $T = (\mathcal{I}, \mathcal{O}, \Delta)$ in a model M, if it ensures that in every run of M in which processes start with an input vector $I \in \mathcal{I}$, there is a finite prefix R of the run in which: (1) decided values form a vector $O \in \mathcal{O}$ such that $(I, O) \in \Delta$, and (2) all correct processes decide. Hence, in the IIS model, all processes must decide.

Simplicial complexes. We use the standard language of *simplicial complexes* [16, 25] to give a combinatorial representation of the IIS model. A *simplicial complex* is defined as a set of *vertices*

and an inclusion-closed set of vertex subsets, called *simplices*. The *dimension* of a simplex σ is $|\sigma| - 1$, and any subset of σ is one of its *faces*. We denote by $\mathbf{s}$ the *standard* $(n-1)$*-simplex*: a fixed set of n vertices and all its subsets.

Given a complex K and a simplex $\sigma \in K$, σ is a *facet* of K, denoted $facet(\sigma, K)$, if σ is not a face of any stricly larger simplex in K. Let $facets(K) = \{\sigma \in K, facet(\sigma, K)\}$. A simplicial complex is *pure* (of dimension n) if all its facets have dimension n. A simplicial complex is *chromatic* if it is equipped with a *coloring function* — a non-collapsing simplicial map χ from its vertices to $\mathbf{s}$, in one-to-one correspondence with n *colors*. In our setting, colors correspond to processes identifiers.

Standard chromatic subdivision and IIS The *standard chromatic subdivision* [18] of a complex K, $\mathrm{Chr}\, K$ ($\mathrm{Chr}\, \mathbf{s}$ is depicted in Figure 1), is a complex where vertices of $\mathrm{Chr}\, K$ are couples (c, σ), where c is a color and σ is a face of K containing a vertex of color c. Simplices of $\mathrm{Chr}\, K$ are the sets of vertices (c_1, σ_1), ..., (c_m, σ_m) associated with distinct colors (i.e., $\forall i, j$, $c_i \neq c_j$) such that the σ_i satisfies the containment and immediacy properties of *IS*. It has been shown that Chr is a *subdivision* [19]. In particular, the geometric realization of $\mathrm{Chr}\, \mathbf{s}$, $|\mathrm{Chr}\, \mathbf{s}|$, is homeomorphic to $|\mathbf{s}|$, the geometric realization of $\mathbf{s}$ (i.e., the convex hull of its vertices). If we *iterate* this subdivision m times, each time applying Chr to all simplices, we obtain the mth chromatic subdivision, Chr^m. $\mathrm{Chr}^m\, \mathbf{s}$ captures the m-round IIS model, IS^m [4, 18].

Given a complex K and a subdivision of it $Sub(K)$, the carrier of a simplex $\sigma \in Sub(K)$ in K, $carrier(\sigma, K)$, is the smallest simplex $\rho \in K$ such that the geometric realization of σ, $|\sigma|$, is contained in $|\rho|$: $|\sigma| \subseteq |\rho|$. The carrier of a vertex $(p, \sigma) \in \mathrm{Chr}\, \mathbf{s}$ is σ. In the matching *IS* task, the carrier corresponds to the snapshot returned by p, i.e., the set of processes *seen* by p. The carrier of a simplex $\rho \in \mathrm{Chr}\, K$ is simply the union (or, due to inclusion, the maximum) of the carriers of vertices in ρ. Given a simplex $\sigma \in \mathrm{Chr}^2\, \mathbf{s}$, $carrier(\sigma, \mathbf{s})$ is equal to $carrier(carrier(\sigma, \mathrm{Chr}\, \mathbf{s}), \mathbf{s})$. $carrier(\sigma, \mathrm{Chr}\, \mathbf{s})$ corresponds to the set of all snapshots seen by processes in $\chi(\sigma)$. Hence, $carrier(\sigma, \mathbf{s})$ corresponds to the union of all these snapshots. Intuitively, it results in the set of all processes *seen* by processes in $\chi(\sigma)$ through the two successive immediate snapshots instances.

Simplex agreement and affine tasks. In the *simplex agreement task*, processes start on vertices of some complex K, forming a simplex $\sigma \in K$, and must output vertices of some subdivision of K, $Sub(K)$, so that outputs form a simplex ρ of $Sub(K)$ respecting carrier inclusion, i.e., $carrier(\rho, K) \subseteq \sigma$. In the simplex agreement tasks considered in the characterization of wait-free task computability [4, 18], K is the standard simplex $\mathbf{s}$ and the subdivision is usually iterations of Chr.

An *affine task* is a generalization of the simplex agreement task, where $\mathbf{s}$ is fixed as the input complex and where the output complex is a pure non-empty sub-complex of some iteration of the standard chromatic subdivision, $\mathrm{Chr}^\ell\, \mathbf{s}$. Formally, let L be a pure non-empty sub-complex of $\mathrm{Chr}^\ell\, \mathbf{s}$ for some $\ell \in \mathbb{N}$. The affine task associated with L is then defined as $(\mathbf{s}, L, \Delta)$, where, for every face $\sigma \subseteq \mathbf{s}$, $\Delta(\sigma) = L \cap \mathrm{Chr}^\ell(\sigma)$. Note that $L \cap \mathrm{Chr}^\ell(\mathbf{t})$ can be empty, in which case the set of participating processes must increase before processes may produce outputs. Note that, since an affine task is characterized by its output complex, with a slight abuse of notation, we use L for both the affine task $(\mathbf{s}, L, \Delta)$ and its ouput complex.

By running m iterations of this task, we obtain L^m, a sub-complex of $\mathrm{Chr}^{\ell m}\, \mathbf{s}$, corresponding to a subset of $IS^{\ell m}$ runs (each of the m iterations includes ℓ *IS* rounds). The affine model associated with L, denoted L^*, corresponds to the set of infinite runs of the IIS model where every prefix

(a) Complex $\mathcal{R}^2_{1-T\&S}$ (b) Complex $\mathcal{R}^2_{2-T\&S}$

Figure 4: $\mathcal{R}^2_{1-T\&S}$ and $\mathcal{R}^2_{2-T\&S}$ in blue.

restricted to a multiple of ℓ *IS* rounds belongs to the subset of $IS^{\ell m}$ runs associated with L^m.

2 Affine task for k-test-and-set

The affine task corresponding to k-test-and-set and read-write shared memory, denoted $\mathcal{R}^{n-1}_{k-T\&S}$, is defined as a subset of the standard chromatic subdivision, $\mathcal{R}^{n-1}_{k-T\&S} \subseteq \mathrm{Chr}(\mathbf{s}^{n-1})$, as follow:

Definition 1

$$\mathcal{R}^{n-1}_{k-T\&S} = \{\sigma \in \mathrm{Chr}(\mathbf{s}^{n-1}) | \forall \sigma' \subseteq \sigma, \forall v, v' \in \sigma', \mathit{carrier}(v) = \mathit{carrier}(v') : |\sigma'| \leq k\}.$$

The definition says that $\mathcal{R}^{n-1}_{k-T\&S}$ includes all simplices of $\mathrm{Chr}(\mathbf{s}^{n-1})$ which does include a face of size greater or equal to k sharing the same carrier.

To see that the definition of $\mathcal{R}^{n-1}_{k-T\&S}$ indeed define an affine task, it must be checked that $\mathcal{R}^{n-1}_{k-T\&S}$ is a pure sub-complex of dimension $n-1$. To see that that any simplex $\sigma \in \mathcal{R}^{n-1}_{k-T\&S}$ is a subset of another simplex $\sigma' \mathcal{R}^{n-1}_{k-T\&S}$ such that $|\sigma'| = n$, one can see that σ can be inductively completed by vertices of missing colors one by one with at each iteration a vertex with a carrier not previously present in the current simplex. The fact that all subset σ' of a simplex $\sigma \in \mathcal{R}^{n-1}_{k-T\&S}$ is also part of $\mathcal{R}^{n-1}_{k-T\&S}$ is a direct corollary of its definition.

This approach would imply to formally prove the intuitively previously stated fact, the advantage of such a definition is that it is easy to show that the complex is k-connected on any restriction on up to k processes as any simplex of size smaller or equal to k is included (or dimension $k-1$). Alternatively it can be put in $\mathcal{R}^{n-1}_{k-T\&S}$ definition that only subsets of $(n-1)$-dimensional simplex of $\mathcal{R}^{n-1}_{k-T\&S}$ are allowed.

3 Solving $\mathcal{R}^n_{k-T\&S}$ with k-Test-and-Set and read-write shared memory.

Solving $\mathcal{R}^n_{k-T\&S}$ using access to shared memory and access to k-Test-and-Set objects is not very difficult. It was previously resolved in [14, 22], the idea consist on using a level-based immediate snapshot implementation which additionally uses access to k-test-and-set objects.

A level-based implementation of an immediate snapshot implementation works as follow: (1) process write their input in memory associated with a level ℓ starting at $\ell = n$; (2) proceed to a

snapshot of memory; (3) if the snapshot contains ℓ values associated with a level $\ell' \leq \ell$ then the process returns with the snapshot consisting of theses ℓ values, otherwise the process re-iterate the process using the level $\ell - 1$. The formal description can be found in Algorithm 1.

Algorithm 1: Level-based immediate snapshot implementation for p_i.

```
Shared Objects: MEM[1...n] ∈ Val × ℕ), initially (⊥, ⊥);
Init: level = n + 1, value = InputValue, snap = ∅;

Do
    level = level − 1;
    MEM[i].Update(value, level);
    snap = MEM.Snapshot();
While |{(v, ℓ) ∈ snap, ℓ ≤ level}| ≠ level;
Return {(v, ℓ) ∈ snap, ℓ ≤ level};
```

Lemma 1 *In an execution of Algorithm 1, at most ℓ processes can reach the $(n-\ell)^{th}$ iteration of the while loop.*

Proof. Let us show this result by induction on the number of iterations k. If $k = 1$, then at most n processes can reach the first iteration of the while loop as there is n processes in the system. Assume that the lemma holds for iteration k, then at most $n-k$ processes may reach iteration k of the while loop. If there is strictly less than k or if a process crash during iteration k, then the lemma property holds for iteration $k+1$. Assume that exactly $n-k+1$ processes reach iteration k and complete this iteration of the while loop and consider the process p which takes last its iteration snapshot. As all $n-k+1$ processes must have previously updated their register with a value associated to level $n-k+1$ and may have been replaced only with the same input value associated to a smaller level value and as at most $n-k+1$ have updated their register with an associated value smaller or equal to $n-k+1$ (i.e., have access the kth iteration of the while loop), then p obtains a snapshot containing exactly $n-k+1$ values associated with a level smaller or equal to $n-k+1$ and thus exit the while loop. Therefore the lemma property holds for level $k+1$ which completes the induction proof. □

Theorem 2 *Algorithm 1 implements an immediate snapshot operation.*

Proof. According to Lemma 1, every correct process eventually exit the while loop and thus returns a set of immediate snapshot input values. The immediate snapshot properties are as follows:

- *self-containment :* A process returns a set of input values including its own. Indeed, a process returning at level ℓ (i.e., with a variable level currently set to ℓ), returns all input values associated to a level smaller or equal to ℓ thus its own as its last update on its register include its input value associated to the level ℓ.
- *containment :* The set of values V and V', returned by two processes p and p' respectively, are such that either $V \subseteq V'$ or $V' \subseteq V$. Consider that p returns at level ℓ and p' returns at level ℓ', w.l.o.g., with $\ell' \leq \ell$. Assume that there is a input value from a process p'' returned

by p' but not by p. p'' must have reached level ℓ' and thus has reached level ℓ, but as its input value is not included by p, p' must have updated its register in iteration corresponding to level ℓ after p proceeded to its snapshot for the same level. But p must have observed ℓ processes which already accessed the iteration corresponding to level ℓ to exit the loop, as it does not include p'' this imply that strictly more than ℓ processes accessed to this iteration, a contradiction with the property of Lemma 1.

- *Immediacy :* Given two processes p and p' returning the set of values V and V' respectively, if V includes the input value from p' then $V' \subseteq V$. To see that the set returned by Algorithm 1 verify this property, assume that this is not the case. By the inclusion property we have that $V \subsetneq V'$. Thus, with ℓ and ℓ' the level at which p and p' returned respectively, we have that $\ell = |V| < |V'| = \ell'$. This implies that p' only updated his register with an associated level greater or equal to ℓ', and so p' input value cannot have been returned by p as only values associated to a level smaller or equal to ℓ are returned, a contradiction.

□

The modification proposed in [14, 22] to solve $\mathcal{R}^n_{k-T\&S}$, consist on modifying the level-based immediate snapshot implementation on its third (conditional) step as follow: (3') if the snapshot contains ℓ values associated with a level $\ell' \leq \ell$ then the process access the k-test-and-set object number ℓ and if it obtains true, then it returns with the snapshot consisting of theses ℓ values, otherwise, in both else conditions, the process re-iterate the process using the level $\ell - 1$.The formal description can be found in Algorithm 2.

Algorithm 2: Solving $\mathcal{R}^n_{k-T\&S}$ for process p_i.

```
Shared Objects: MEM[1...n] ∈ Val × ℕ), initially (⊥, ⊥);
Init: level = n + 1, exit = true, value = InputValue, snap = ∅;

Do
    Do
        level = level − 1;
        MEM[i].Update(value, level);
        snap = MEM.Snapshot();
    While |{(v, ℓ) ∈ snap, ℓ ≤ level}| ≠ level;
    exit = k-Test&Set[level];
While ¬exit;
Return {(v, ℓ) ∈ snap, ℓ ≤ level};
```

Lemma 3 *In an execution of Algorithm 2, at most ℓ processes can reach the $(n-\ell)^{th}$ iteration of the inner while loop.*

Proof. Let us show this result by induction on the number of iterations k of the inner while loop. If $k = 1$, the claim is trivially verified. Let us assume that the lemma holds for iteration k, then at most $n - k$ processes may reach iteration k of the while loop. If there is strictly less than k or if a process crash during iteration k, then the lemma property holds for iteration $k + 1$. So

let us now assume furthermore that exactly $n-k+1$ processes reach iteration k and complete it without crashing, and consider the process p which takes last its iteration snapshot. As all $n-k+1$ processes must have previously updated their register with a value associated to level $n-k+1$ and may have been replaced only with the same input value associated to a smaller level value and as at most $n-k+1$ have updated their register with an associated value smaller or equal to $n-k+1$ (i.e., have access the kth iteration of the while loop), then p obtains a snapshot containing exactly $n-k+1$ values associated with a level smaller or equal to $n-k+1$ and thus exit the inner while loop.

If p the obtains true as output to the accessed k-test-and-set object, then it exit the second while loop and returns from the algorithm. Otherwise, if p obtains false, this implies that another process accessed the k-test-and-set object and, if it does not crash, received true. Therefore as this process can only be one of the $n-k+1$ which accessed the kth iteration, one less process participate and therefore the lemma property holds for level $k+1$, which completes the induction proof. □

Theorem 4 *Algorithm 2 solves* $\mathcal{R}^n_{k-T\&S}$.

Proof. First let us show that Algorithm 2 solves the immediate snapshot task. It can be noticed that the proof of Theorem 2 still applies to the modified version of Algorithm 2, only Lemma 1 proof is no longer, but Lemma 3 may be used as replacement.

As Algorithm 2 solves the immediate snapshot task, all left to be shown is that at most k process output at the same level, i.e., are provided with the same output set and thus are associated to vertices on the same carrier. It is trivial to see that this is the case. Indeed, consider a set of processes Q all outputting on the same level ℓ, they therefore all accessed the same k-Test-and-Set object (associated to level ℓ) and all obtained true as object output. As at most k processes may obtain the output true from a k-Test-and-Set object, then Q is of a size of at most k. □

4 Simulating k-Test-and-Set and read-write shared memory in $\mathcal{R}^n_{k-T\&S}{}^*$.

Simulating k-Test-and-Set and read-write shared memory in $\mathcal{R}^n_{k-T\&S}{}^*$ is rather simple. The idea is to use the restriction on the number of process with the same immediate snapshot output to solve access to k-Test-and-Set by simulating a write operation. Read/write simulations using iterated (immediate) snapshots are quite common in the literature and any one of them can be used for our purpose. With their simulated write values, processes add a tag which can correspond to a classic write or to a k-test-and-set operation where the write content is replaced with the identifier of the corresponding accessed simulated object. To decide what to return on a completed simulated operation corresponding to a k-test-and-set operation, processes check whether the returned snapshot contains more than k accesses to the corresponding k-test-and-set object and if there is less or equal to k accesses they return true and otherwise false.

By using a full information protocol which associate all past operations to any simulated operations, this simple scheme ensure that at most k processes obtains a true as output to an access to a given k-test-and-set object. The structure of $\mathcal{R}^n_{k-T\&S}$ ensure that simulated snapshots contains

at most k new inputs not returned in a previous round and therefore that at least one process accessing a given k-test-and-set object returns true.

5 Solving $\mathcal{R}^n_{k-T\&S}$ in the $\{(k+1,k)\}$-model.

Solving $\mathcal{R}^n_{k-T\&S}$ in the $\{(k+1,k)\}$-model directly is more difficult than using the equivalent read/write shared memory model with access to k-test-and-set objects. The idea is to run an adaptive BGG-simulation where by batches, k simulators first give output to processes on a set of $k+1$ levels. More precisely, BG-simulators from range mk to $(m+1)k-1$ simulate processes, in a depth first manner, up to level $m(k+1)$ and ignoring first the output test on levels higher or equal to $(m+1)(k+1)$. Every time a process is provided with an output, then the range are updated by shifting the target range for each terminated process on a smaller level or on a level included in the target range.

6 Simulating the $\{(k+1,k)\}$-model in $\mathcal{R}^n_{k-T\&S}{}^*$.

Simulating the $\{(k+1,k)\}$-model in $\mathcal{R}^n_{k-T\&S}{}^*$ directly is not much difficult than simulating shared memory and access to k-test-and-set objects according that at most $k+1$ processes may access the same object. The idea is still to use simulated write operation for agreement operation, instead of returning true, a process returns its own proposal and instead of returning false, a process returns the proposal from any other process which accessed the object in previous rounds.

7 Related work

Herlihy and Shavit [18] proposed a characterization of wait-free task computability through the existence of a simplicial map from a subdivision of the input complex of a task $\mathcal{I}$ to its output complex $\mathcal{O}$. (The reader is referred to [16] for a thorough discussion of the use of combinatorial topology in distributed computability.) Herlihy and Rajsbaum [17] studied colorless task computability in the special case of *superset-closed* adversaries. They show that the protocol complex of a superset-closed adversary with *minimal core size* c is $(c-2)$-connected. This result, obtained via an iterative application of the Nerve lemma, gives a combinatorial characterization of superset-closed adversaries. The characterization only applies to colorless tasks, and it does not allow us to express the adversary in an affine way.

Gafni et al. [13] introduced the notion of an affine task and characterized task computability in *iterated* adversarial models via infinite subdivisions of input complexes, assuming a limited notion of solvability that only guarantees outputs to "fast" processes [5,10] (i.e., "seen" by every other process infinitely often). The liveness property defined in this paper for iterated models guarantees outputs for *every* process, which allowed us to establish a task-computability equivalence with conventional non-iterated models.

Saraph et al. [24] gave a compact combinatorial characterization of t-resilient task computability. Note that $\mathcal{A}_{t-res}$ is a superset-closed (and thus fair) adversary. Our solution of the affine task $\mathcal{R}_{\mathcal{A}}$ in the α-model is inspired by the t-resilient solution of $\mathcal{R}_{t-res}$ in [24]. Gafni et al. [11] presented affine tasks for the model of k-set consensus and, thus, k-concurrency and k-obstruction-freedom, which can be expressed as a symmetric and thus fair adversary.

The notions of agreement functions and a fair adversaries were introduced by the first two authors in [21]. One can determine the agreement function of any given adversary using the formula suggested earlier for the set consensus power [12]. It has been shown in [21] that agreement functions encode enough information to characterize the task computability of any fair adversary.

8 Concluding remarks

This paper propose a new affine characterization of the wait-free shared memory model enhanced with k-test-and-set objects. Just as the wait-free characterization [18] implies that the *IS* task captures the wait-free model, our characterization equates any such model with a (compact) affine task embedded in the standard chromatic subdivision.

Interestingly, unlike [24], we cannot rely on the *shellability* [16] (and, thus, link-connectivity) of the affine task. Link-connectivity of a simplicial complex $\mathcal{C}$ allows us to work in the *point set* of its geometrical embedding $|\mathcal{C}|$ and use continuous maps (as opposed to simplicial maps that maintain the simplicial structure). For example, the existence of a continuous map from $|\mathcal{R}_{\mathcal{A}_{t-res}}|$ to any $|\mathcal{R}^k_{\mathcal{A}_{t-res}}|$ implies that $\mathcal{R}_{\mathcal{A}_{t-res}}$ indeed captures the general task computability of $\mathcal{A}_{t-res}$ [24]. In general, however, the existence of a continuous map onto $\mathcal{C}$ only allows us to converge on a *single* vertex [16]. If $\mathcal{C}$ is not link-connected, converging on one vertex allows us to compute an output only for a single process, and not more. Unfortunately, only very special adversaries, such as $\mathcal{A}_{t-res}$, have link-connected counterparts (see, e.g., the affine task corresponding to 1-test-and-set in Figure 4a). Instead of relying on link-connectivity, this paper takes an explicit algorithmic way of showing that iterations of $\mathcal{R}_{k-T\&S}$ simulate the wait-free shared memory model enhanced with k-test-and-set objects. An interesting question is to which extent point-set topology and continuous maps can be applied in affine characterizations.

Nailed down, this may allow us to compactly capture all "natural" models [11], such as, e.g., generic adversarial models or the *set consensus collections* models [7] for which only special cases of k-set consensus [11] and k-test-and-set have been, in this sense, understood so far.

References

[1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873–890, 1993.

[2] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC*, pages 91–100. ACM Press, May 1993.

[3] Elizabeth Borowsky and Eli Gafni. Immediate atomic snapshots and fast renaming. In *PODC*, pages 41–51, New York, NY, USA, 1993. ACM Press.

[4] Elizabeth Borowsky and Eli 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*, pages 189–198, New York, NY, USA, 1997. ACM Press.

[5] Zohir Bouzid, Eli Gafni, and Petr Kuznetsov. Strong equivalence relations for iterated models. In *OPODIS*, pages 139–154, 2014.

[6] Soma Chaudhuri. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, pages 311–324, Québec City, Québec, Canada, August 1990.

[7] Carole Delporte-Gallet, Hugues Fauconnier, Eli Gafni, and Petr Kuznetsov:. Set-consensus collections are decidable. In *OPODIS*, 2016.

[8] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Andreas Tielmann. The disagreement power of an adversary. *Distributed Computing*, 24(3-4):137–147, 2011.

[9] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[10] Eli Gafni. Round-by-round fault detectors (extended abstract): Unifying synchrony and asynchrony. In *Proceedings of the 17th Symposium on Principles of Distributed Computing*, 1998.

[11] Eli Gafni, Yuan He, Petr Kuznetsov, and Thibault Rieutord. Read-write memory and k-set consensus as an affine task. In *OPODIS*, 2016. Technical report: https://arxiv.org/abs/1610.01423.

[12] Eli Gafni and Petr Kuznetsov. Turning adversaries into friends: Simplified, made constructive, and extended. In *OPODIS*, pages 380–394, 2010.

[13] Eli Gafni, Petr Kuznetsov, and Ciprian Manolescu. A generalized asynchronous computability theorem. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 222–231, 2014.

[14] Eli Gafni, Michel Raynal, and Corentin Travers. Test & set, adaptive renaming and set agreement: A guided visit to asynchronous computability. In *SRDS*, pages 93–102, 2007.

[15] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):123–149, January 1991.

[16] Maurice Herlihy, Dmitry N. Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2014.

[17] Maurice Herlihy and Sergio Rajsbaum. Simulations and reductions for colorless tasks. In *PODC*, pages 253–260, 2012.

[18] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(2):858–923, 1999.

[19] Dmitry N. Kozlov. Chromatic subdivision of a simplicial complex. *Homology, Homotopy and Applications*, 14(1):1–13, 2012.

[20] Petr Kuznetsov. Understanding non-uniform failure models. *Bulletin of the EATCS*, 106:53–77, 2012.

[21] Petr Kuznetsov and Thibault Rieutord. Agreement functions for distributed computing models. In *NETYS*, pages 175–190, 2017.

[22] Achour Mostéfaoui, Michel Raynal, and Corentin Travers. Exploring gafni's reduction land: From *mega* to wait-free adaptive (2p-[p/k])-renaming via k-set agreement. In *DISC*, pages 1–15, 2006.

[23] Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. on Computing*, 29:1449–1483, 2000.

[24] Vikram Saraph, Maurice Herlihy, and Eli Gafni. Asynchronous computability theorems for t-resilient systems. In *DISC*, pages 428–441, 2016.

[25] Edwin H. Spanier. *Algebraic topology.* McGraw-Hill Book Co., New York, 1966.

[26] Gadi Taubenfeld. The computational structure of progress conditions. In *DISC*, 2010.