Affine Tasks for k-Test-and-Set

Petr Kuznetsov, Thibault Rieutord

Affine Tasks for k-Test-and-Set

Petr Kuznetsov[1] and Thibault Rieutord[2]

[1] LTCI, Télécom Paris, Institut Polytechnique Paris

[2] Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Abstract

The paper proposes a surprisingly simple characterization of task computability of the wait-free shared-memory model in which processes, in addition to read-write registers, have access to *k-test-and-set* objects. Our characterization is expressed in the form of an *affine task*: a subcomplex of some iteration of the standard chromatic subdivision. This appears to be the first topological characterization of a model in which processes communicate via long-lived objects beyond read-write registers.

1 Introduction

One of the central challenges in the theory of distributed computing is determining *relative computability* of its numerous models, parameterized by types of failures they expose (crash, omission, Byzantine), synchrony hypotheses they assume (asynchronous, partially synchronous, synchronous), and communication primitives they employ (message-passing, read-write registers, powerful shared objects). Starting from the seminal work by Herlihy and Shavit [18], task computability of multiple models of computation have been characterized using the language of combinatorial topology [12, 17, 22, 25]. More precisely, given a task T and a model of computation M, we can equate the question of whether T is solvable in M with the existence of a specific continuous map between a transformed *input complex* of T and the *output complex* of T, *carried by* T, i.e., preserving the task specification. M entirely determines the way the input complex is transformed.

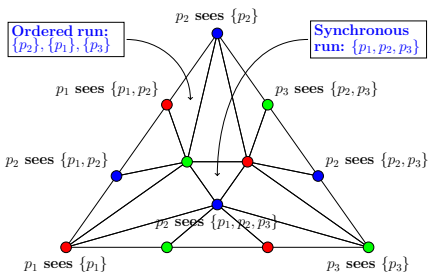

Figure 1: Chr(**s**), the standard chromatic subdivision of a 2-simplex, the output complex of the 3-process *IS* task.

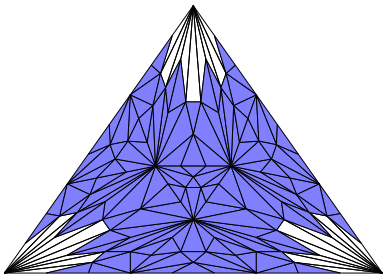

Figure 2: $\mathcal{R}_{1-res}$, the affine task of 1-resilience (in blue).

For example, to characterize task computability in the *wait-free* read-write model [15], we can simply consider a *subdivision* of the input complex [18]. In particular, we can choose this subdivision to be a number of iterations of the *standard chromatic* subdivision (denoted Chr, Figure 1). The complex captures one round of *immediate snapshot* (IS) [3].

Task computability in the t-resilient read-write model has been characterized [25] via a specific task $\mathcal{R}_{t-res}$. The task is defined for n processes 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 iteration 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. Figure 2 depicts the output complex of $\mathcal{R}_{1-res}$, the affine task for the 3-process 1-resilient model.

Complex Chr and $\mathcal{R}_{t-res}$ are called *affine tasks* [11, 12] for for the wait-free model and t-resilient model, respectively. More generally, an affine task $\mathcal{R}_M$ for a model M is defined as a subcomplex of a finite number of iteration of the standard chromatic subdivision, such that a task T is solvable in M if and only if there exists a continuous map from a finite number of iterations of $\mathcal{R}_M$ on the input complex of T to the output complex of T, carried by T.

Recently, affine tasks for a large class of *fair adversarial* models have been characterized via affine tasks [22]. An adversarial [8] shared-memory is defined via 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 $\mathcal{A}$. Computability of adversarial models has been characterized for several special cases: wait-freedom [18], t-resilience [25], k-concurrency [11] and, finally, fair adversaries [22].

These characterizations mostly focused on *read-write* shared memory. The only exception is the work by Gafni et al. [11], where the processes could, additionally, access k-set consensus objects. Via simulations, this model was shown to be equivalent to the model of *k-concurrency* that assumes up to k processes are allowed to be concurrently active. However, to the best of our knowledge, no direct topological characterization has been proposed for models in which processes communicate via powerful shared objects, other than read-write registers. In this paper, we complement earlier results with a simple characterization of a model in which processes, in addition to read-write registers, can access k-test-and-set objects, for a fixed natural k. A k-test-and-set object is accessed by a single operation that eventually returns either 1 or 0, so that at least 1 and at most k of the participating processes obtain 1.

It has been observed that computability of a model in distributed computing is tightly coupled with its ability to solve *set consensus*. The *agreement function* [20, 24] of a model

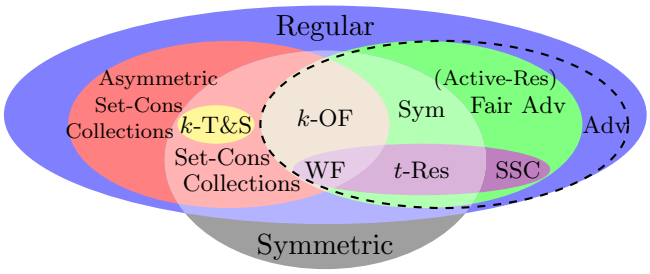

Figure 3: A classification of shared-memory models based on agreement functions.

specifies the "best" level of set consensus, i.e., the minimal guaranteed number of distinct output values, for all sets of *participating* (proposing inputs and expecting outputs) and *active* (taking steps) processes. Depending on their agreement functions, models can be classified in (1) *symmetric* models, where agreement functions only depend on the set cardinalities, (2) *fair*, where agreement functions do not depend on the active sets, (3) *local*, where agreement functions do not depend on the participating sets, and (4) *regular*, where agreement function increase with increasing participation. In Figure 3, we depict interrelations between these and other shared-memory models.

In this paper, we define an affine task $\mathcal{R}_{k-T\&S}$ capturing the task computability of the wait-free read-write shared memory models equipped 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.

Task $\mathcal{R}_{k-T\&S}$ (see Figure 5) can be defined as a subcomplex of Chr, just a single iteration of the standard chromatic subdivision. In contrast, affine tasks of other models, such as k-concurrency or t-resilience ($1 < k < n$ and $0 < t < n-1$) require at least two iterations [7,11,25].

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. Affine tasks may also lead to decidable characterization of relative task computability, as has been recently shown for 2-process models [21].

Roadmap. Section 2 reviews our model definitions. Section 3 defines our affine task $\mathcal{R}_{k-T\&S}$. In Section 4, 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 5, 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 6 reviews related work and concludes the paper.

2 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].

Figure 4: Examples of valid sets of IS outputs. On the left, we have the "ordered" execution in which every process outputs a distinct set of inputs (blue - only itself, red - blue and itself, and green - all three). On the right, we have the "synchronous" execution in which all three processes output all the inputs.

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*.

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 always 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 4 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.

k-test-and-set model. For an integer $k \geq 1$, a *k-test-and-set* object exports one operation, *apply*(), that may be only accessed once by each process, takes no parameters, and returns a boolean value. It guarantees that at most k processes will get 1 as output and that not all processes accessing it obtained 0. In the special case of $k = 1$ the object is simply called *test-and-set*.

The *k-test-and-set model* is then simply defined as the wait-free model with, additionally, access to any number of k-test-and-set objects. Hence processes run a full-information protocol on an AS memory without any restrictions on the set of possible runs, but processes may proceed, between any operations on the AS memory, to operations on any number of k-test-and-set objects.

Simplicial complexes. We use the standard language of *simplicial complexes* [16, 26] 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 the number of vertices of σ minus one, 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 strictly larger simplex in K. Let $facets(K) = \{\sigma \in K, facet(\sigma, K)\}$. A simplicial complex is *pure* (of dimension m) if all its facets have dimension m.

A map α from the vertices of a complex K to the vertices of a complex L is *simplicial* if each simplex in K is mapped to a simplex in L. A simplicial map $\alpha : K \to L$ is *rigid* if for all $\sigma \in K$, $|\sigma| = |\alpha(\sigma)|$.

A simplicial complex is *chromatic* if it is equipped with a *coloring function* — a rigid 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*.

Every simplex σ has a *geometric realization* $|\sigma|$. It is obtained by representing the vertices of σ as an affinely independent set of points in a Euclidean space and then taking the convex

hull of them. A geometric realization of a complex K, denoted by $|K|$, is the union of geometric realizations of its simplices, properly "glued" along their faces [16].

It has been shown that Chr is a *subdivision* [19], i.e., informally, $|\operatorname{Chr}\mathbf{s}|$ is homeomorphic to $|\mathbf{s}|$. If we *iterate* this subdivision m times, each time applying Chr to all simplices, we obtain the m^{th} chromatic subdivision, $\operatorname{Chr}^m\mathbf{s}$. $\operatorname{Chr}^m\mathbf{s}$ precisely captures the runs of 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 $(p, \sigma) \in \operatorname{Chr}\mathbf{s}$ 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 $\rho \in \operatorname{Chr} K$ is simply the union (or, due to inclusion, the maximum) of the carriers of vertices in ρ. Given a simplex $\sigma \in \operatorname{Chr}^2\mathbf{s}$, $carrier(\sigma, \mathbf{s})$ is equal to $carrier(carrier(\sigma, \operatorname{Chr}\mathbf{s}), \mathbf{s})$. $carrier(\sigma, \operatorname{Chr}\mathbf{s})$ 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, $\operatorname{Chr}^\ell\mathbf{s}$. Formally, let L be a pure non-empty sub-complex of $\operatorname{Chr}^\ell\mathbf{s}$ for some $\ell \in \mathbb{N}$. The affine task associated with L is then defined as $(\mathbf{s}, L, \Delta)$, where, for every face $\sigma \subseteq \mathbf{s}$, $\Delta(\sigma) = L \cap \operatorname{Chr}^\ell(\sigma)$. Note that $L \cap \operatorname{Chr}^\ell(\mathbf{t})$ 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 $\operatorname{Chr}^{\ell m}\mathbf{s}$, 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 restricted to a multiple of ℓ *IS* rounds belongs to the subset of $IS^{\ell m}$ runs associated with L^m.

3 Affine task for k-test-and-set

In this section, we define affine task $\mathcal{R}_{k-T\&S}$ capturing computability of the k-test-and-set model. The task is defined as a simple subcomplex of a single iteration of the standards chromatic subdivision.

The intuition is the following. Test-and-set solves, in a straightforward manner, perfect renaming [2]. It also provides adaptive solutions to renaming in which *names* of the processes reflect the order in which they access the task. Hence a process obtaining the name j can see all values shared previously by processes that receive a smaller name i with $i < j$. It can

also be used to solve immediate snapshot [3] in a way that every process obtains a distinct rank j and observes the inputs of all processes with smaller positions: the process of level j sees precisely j inputs, its own plus those of the processes with strictly lower ranks. Such an immediate snapshot execution essentially impose *total order* on the processes.

We can naturally generalize this observation to k-test-and-set objects. Indeed, consider a partial order in which every process is associated with a rank so that at most k processes share the same rank. Similarly, a process can observe the values previously shared by processes obtaining a lower or equal rank. In an immediate snapshot, this is equivalent to having at most k processes sharing the same output. It leads to a definition of *k-ordered* executions or corresponding simplices of Chr: among any set of $k+1$ processes, at least two have different ranks. Note that total order executions corresponding to 1-test-and-set are 1-ordered.

$\mathcal{R}_{k-T\&S}$ captures the set of k-ordered executions. Formally, $\mathcal{R}_{k-T\&S}$ is the set of simplices of $\mathrm{Chr}\,\mathbf{s}$, the standard chromatic subdivision, in which at most k vertices share the same carrier:

Definition 1. *$\mathcal{R}_{k-T\&S}$ is equal to:*

$$\sigma \in \mathrm{Chr}(\mathbf{s}) : \forall \sigma' \subseteq \sigma, (\forall v, v' \in \sigma', \mathit{carrier}(v) = \mathit{carrier}(v')) \implies |\sigma'| \leq k.$$

To be an affine task, $\mathcal{R}_{k-T\&S}$ needs to be a pure sub-complex of $\mathrm{Chr}\,\mathbf{s}$ of the same dimension:

Property 1. *$\mathcal{R}_{k-T\&S}$ is an affine task.*

Proof. The fact that $\mathcal{R}_{k-T\&S}$ is a sub-complex of $\mathrm{Chr}\,\mathbf{s}$ is trivial. Indeed, the definition is inclusion-closed. Consider any simplex $\sigma \in \mathcal{R}_{k-T\&S}$ and any face of it $\sigma' \subseteq \sigma$. Any face of σ' is a face of σ and hence satisfies the condition of having at most k vertices sharing the same carrier.

Showing that $\mathcal{R}_{k-T\&S} \subseteq \mathrm{Chr}\,\mathbf{s}$ is pure and of the same dimension as $\mathbf{s}$ is less trivial. For this, we need to show that any simplex $\sigma \in \mathcal{R}_{k-T\&S}$ is a face of a simplex $\sigma' \in \mathcal{R}_{k-T\&S}$ of dimension equal to $\mathit{dim}(\mathbf{s})$. Note that by transitivity, it is sufficient to show that any simplex of $\mathcal{R}_{k-T\&S}$ of a strictly smaller dimension than $\mathit{dim}(\mathbf{s})$ is the face of a strictly larger simplex of $\mathcal{R}_{k-T\&S}$.

Consider a simplex $\sigma \in \mathcal{R}_{k-T\&S}$ and any color c from Π such that $c \notin \chi(\sigma)$ and let $v \in \mathbf{s}$ be the vertex of $\mathbf{s}$ of color c. Two cases may happen: either c is a color of the carriers every vertex of in σ, or else, there exists a vertex in σ with the largest carrier $\mathbf{t}$ such that $c \notin \chi(\mathbf{t})$. In the former case, we can add vertex $(c, \{v\})$ to σ. In the latter case, we can add the vertex $(c, \{v\} \cup \mathbf{t})$ to σ. It is easy to check that the new simplex still verifies the immediacy, self-inclusion, and containment properties and, thus, belongs to $\mathrm{Chr}\,\mathbf{s}$. Moreover, the carrier of the new vertex is distinct (shared by no vertex in σ), and hence the new simplex belongs to $\mathcal{R}_{k-T\&S}$. Indeed, any vertex $v \in \sigma$ such that $\chi(v) \in \chi(\mathbf{t})$ has a carrier that is a face of $\mathbf{t}$ due to the immediacy property. Hence, as long as there are missing colors, we can find a larger simplex in $\mathcal{R}_{k-T\&S}$ including σ as a face. Hence, $\mathcal{R}_{k-T\&S} \subseteq \mathrm{Chr}\,\mathbf{s}$ is indeed a pure complex of dimension of $\mathbf{s}$. □ □

The affine tasks corresponding to 3-process models of 1-test-and-set and 2-test-and-set are depicted in Figure 5. Note that affine tasks' facets are displayed in blue and, thus, the faces of blue simplices also belong to the affine task.

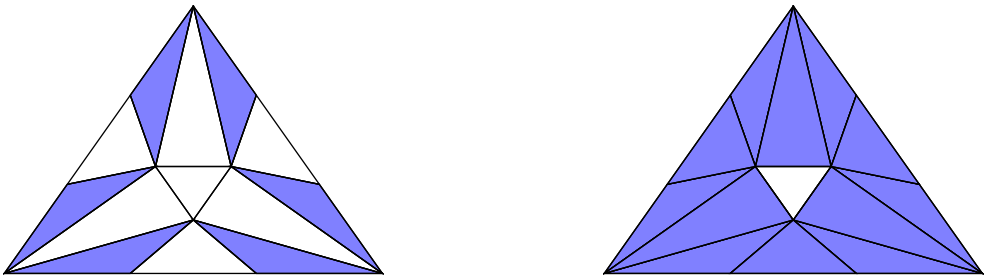

Figure 5: 3-process affine tasks $\mathcal{R}_{1-T\&S}$ and $\mathcal{R}_{2-T\&S}$ with their facets displayed in blue.

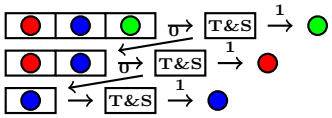

Figure 6: Ordered IS algorithm with test-and-set.

4 Solving $\mathcal{R}_{k-T\&S}$ in the k-test-and-set model

Solving $\mathcal{R}_{k-T\&S}$ using k-test-and-set objects and read-write registers is rather straightforward. The idea, originally suggested in [14, 23], consists in using a level-based immediate snapshot implementation that additionally uses access to k-test-and-set objects.

Recall that the level-based implementation of an immediate snapshot [3] operates as follows. Starting with level $\ell = n$, every process (1) writes its input and ℓ in the memory array; (2) takes a snapshot of the array; (3) if the snapshot contains ℓ values associated with levels $\ell' \leq \ell$, the process returns the snapshot consisting of these ℓ values; otherwise, the process proceeds to level $\ell - 1$ (see Figure 4).

The modification proposed in [14, 23] to solve $\mathcal{R}_{k-T\&S}$, consists in modifying step (3) of this implementation as follows: (3') if the snapshot contains ℓ values associated with a level $\ell' \leq \ell$ then processes accesses the k-test-and-set object number ℓ. If the k-test-and-set object returns *true*, then the process terminates with the snapshot consisting of these ℓ values; otherwise (in both else conditions), the process proceeds to level $\ell - 1$. The formal description is depicted in Algorithm 1, Figure 6 illustrates the general procedure.

Theorem 2. *Algorithm 1 solves $\mathcal{R}_{k-T\&S}$.*

This result implies that every task solvable in $\mathcal{R}^*_{k-T\&S}$ is solvable in the k-test-and-set model. Indeed, a task solvable in $\mathcal{R}^*_{k-T\&S}$ implies a solution ϕ from $\mathcal{R}^m_{k-T\&S}(\mathcal{I})$ to $\mathcal{O}$, for some given m. One can thus simply iterate the solution of Algorithm 1 m times using its input and return the solution provided by ϕ to obtain a task solution in the k-test-and-set model.

Let us show two auxiliary lemmas before presenting the proof of Theorem 2.

Lemma 3. *In an execution of Algorithm 1, at most $n - \ell + 1$ processes can reach the ℓ^{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 $\ell = 1$, the claim is trivially verified. Let us assume that the lemma holds for iteration ℓ, then at most $n - \ell + 1$ processes may reach iteration ℓ of the while loop. If there is strictly

Algorithm 1: Solving $\mathcal{R}_{k-T\&S}$ for process p_i.

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

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

less than ℓ or if a process crash during iteration ℓ, then the lemma property holds for iteration $\ell + 1$. So let us now assume furthermore that exactly $n - \ell + 1$ processes reach iteration ℓ and complete it without crashing. Consider the process p that takes last its snapshot for this iteration. All these $n - \ell + 1$ processes must have previously updated their register with a value associated with level $n - \ell + 1$ and may have replaced it only with the same input value associated with a smaller level value. Thus, p obtains a snapshot containing exactly $n - \ell + 1$ values associated with a level lower than or equal to $n - k + 1$ and consequently exit the inner while loop.

If p the obtains true as output to the accessed k-test-and-set object, then it exits 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 $n - \ell + 1$ that accessed the ℓ^{th} iteration, one less process participates. Thus, the lemma property holds for level $\ell + 1$, which completes the induction proof. □

Lemma 4. *Algorithm 1 implements an immediate snapshot operation.*

Proof. According to Lemma 3, 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 with a level smaller or equal to ℓ thus its own as its last update on its register include its input value associated with 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 the 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 3.

- *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 $\ell = |V| < |V'| = \ell'$. It implies that p' only updated his register with an associated level greater or equal to ℓ'. So p' input value cannot have been returned by p as only values associated with a level smaller or equal to ℓ are returned — a contradiction.

□

Let us now go back to the proof of Theorem 2:

Proof. As Algorithm 1 solves the immediate snapshot task (Lemma 4), all left to be shown is that at most k process output at the same level, i.e., is provided with the same output set and thus are associated with vertices on the same carrier. It is trivial to see that this is the case. Consider a set of processes Q that all output on the same level ℓ. Therefore, they all accessed the same k-Test-and-Set object (associated with level ℓ), and all obtained true as object output. As at most k processes may get the output true from a k-Test-and-Set object, then Q is of a size of at most k. □

5 Simulating k-test-and-set model in $\mathcal{R}^*_{k-T\&S}$

Simulating the k-test-and-set model using iterations of $\mathcal{R}_{k-T\&S}$ ($\mathcal{R}^*_{k-T\&S}$) is slightly less straightforward. Simulating shared memory is well known for iterations of the standard chromatic subdivision, hence, for a subset of such runs as well. A standard simulation described in [13] (for completeness described in the appendix A) ensures progress to the non-terminated processes with infinitely often the smallest view. This provides lock-freedom, one process makes progress and will eventually return with a task output. Hence, this is enough to ensure that eventually all processes can obtain a task output. Our goal is to ensure that the same set of "fast" processes make progress with their k-test-and-set operations as well.

As not all processes may participate in a k-test-and-set operation, processes with the smallest view need to progress independently. But processes with a larger view could later participate in a round in which they have the smallest view. Hence, to ensure that not more than k processes return 1, slow processes must preemptively fail the test-and-set operations. Unfortunately, this is not possible, as a process can only identify processes with smaller views and not precisely those that have the smallest view. Indeed, we could have a process preemptively fail without anyone returning 1. We resolve this issue by simulating k-set-consensus operations among sets of $k+1$ processes and showing then that it is sufficient to simulate n-process k-test-and-set operations.

Let us first show how a $k+1$-process k-test-and-set can be used to implement an n-process k-test-and-set. Then, we will show how k-set consensus among $k+1$ processes can be used to implement k-test-and-set among $k+1$ processes. Lastly, we are going to show how to simulate operations of k-set consensus among $k+1$ processes in $\mathcal{R}^*_{k-T\&S}$.

From $k+1$-process test-and-set to n-process k-test-and-set. In the solution, depicted in Algorithm 2, every process participates in k-test-and-set operations among any possible subset of $k+1$ processes it belongs to. The processes iterate on over these sets of $k+1$ processes in the same deterministic order and return 0 as soon as they obtain 0 from a k-test-and-set operation. If a process manages to obtain 1 from all k-test-and-set operations, it returns 1.

Algorithm 2: n-process $k-T\&S$ using $k+1$-process k-$T\&S$ for p_i.

```
1 forall S ⊆ Π, |S| = k + 1 do
2 |   if i ∈ S then
3 |   |   if k − T&S[S].apply()=0 then Return 0 ;
4 Return 1;
```

Theorem 5. *Algorithm 2 solves n-process k-test-and-set.*

Proof. Let us first show that at most k process may return 1. It is straightforward. Indeed, assume that there are $k+1$ processes returning 1. They must have all accessed the same k-test-and-set operation corresponding to their set of $k+1$ processes. But at most k of them may have obtained 1 from it, the remaining ones must have therefore returned from the protocol with 0 — a contradiction.

Let us now show that not all participating processes may return 0. Indeed, consider the last set of $k+1$ processes in the sequence for which the associated k-test-and-set object has been accessed. Not all processes accessing this object can return 0. Hence, they must either access another k-test-and-set object afterward, which is not the case by assumption or return with 1 or crash. Therefore, not all participating processes can return 0. □ □

From $(k+1)$-process k-set consensus to k-test-and-set. Using k-set consensus among $k+1$ processes to solve k-test-and-set operations among $k+1$ processes is straightforward. Processes can access a k-set consensus operation with their identifier. Then they write their output to the shared-memory and take a snapshot. If a process sees that some process obtained its identifier as output, it returns 1, and otherwise, it returns 0. See Algorithm 3 for a formal description.

Algorithm 3: $k+1$-process $k-T\&S$ using $k+1$-process k-set-consensus for p_i.

```
1 res = k-set-consensus[i];
2 MEM.update(res);
3 snap = MEM.snapshot();
4 if i ∈ snap then
5 |   Return 1;
6 else
7 |   Return 0;
```

Theorem 6. *Algorithm 3 solves $k+1$-process k-test-and-set.*

Proof. A process can return with 1 only if its identifier was returned to some process, hence at most k process can obtain 1. Assume now that all participating processes terminate and

consider the process for which its identifier was first written to the shared memory. This process must see its identifier in its snapshot and return 1, hence not all participating processes may return 0. □ □

Solving k-set-consensus among $k+1$ processes in $\mathcal{R}^*_{k-T\&S}$. The advantage of k-set consensus operations compared to k-test-and-set operations is that processes can participate as soon as they see some process participating. Indeed, since it is a colorless task, processes can adopt inputs from any other process. Hence, to solve k-set consensus operations among $k+1$ processes, processes maintain a decision estimate for all k-set consensus operations and share them in all iterations of $\mathcal{R}^*_{k-T\&S}$. When a process initiates a new operation for which it has no decision estimate yet, it simply adds a decision estimate corresponding to its input value. Moreover, when a process sees a process participating in a new operation, it adopts its decision estimate.

Now, at the end of each iteration of $\mathcal{R}_{k-T\&S}$, processes look at the decision estimate for all operations. If a process sees all $k+1$ potential participants of an operation, then it replaces its decision estimate by the decision estimate of the process with the next identifier (going back to the first to form a loop when there are none higher). For a process to terminate, it must see that all potential participants share a decision estimate for the same round. If it happens, processes return their potentially updated decision estimate as k-set consensus output.

Note that once terminated, processes use a special input value $\perp$. When a process competes with a terminated process for a k-set consensus operation, then it directly returns with its proposal.

Correctness of the simulation of k-set consensus among $k+1$ processes. Let us first show that simulated operations respect the specification of k-set consensus among $k+1$ processes before showing that sufficient progress is also guaranteed.

Lemma 7. *The simulation satisfies the safety properties of k-set consensus among $k+1$ processes in $\mathcal{R}^*_{k-T\&S}$.*

Proof. Processes return their decision estimate which is initially set to their input or adopted from other processes decision estimates. Hence validity is satisfied.

Now consider the first iteration of the affine task after which the first process returns with an output. In this iteration, all processes with the smallest view shared a decision estimate. Hence, all processes adopted a decision estimate at the end of the round. If at the end of the round there are less than k distinct decision estimates, then the agreement property will be ensured as the number of distinct decision estimates in later rounds is a subset of this one.

To see that there are at most k distinct decision estimates at the end of this first iteration in which a process decides, consider the processes which see the $k+1$ potential participants. These processes adopt the decision estimate of the next process (relatively to identifier ranks). But in $\mathcal{R}_{k-T\&S}$, at most k vertices may share the same carrier. Hence a process seeing all participants must adopt the decision estimate of a process not seeing all of them. But this process does not change its decision estimate. Thus, two processes share the same decision estimate. The number of distinct decision estimates is, therefore, smaller than or equal to k and hence at most k distinct outputs may be returned. □ □

Lemma 8. *The simulation of k-set consensus among $k+1$ in $\mathcal{R}^*_{k-T\&S}$ provides progress to processes having infinitely often the smallest view among non-terminated processes.*

Proof. Processes participate in an operation as soon as they see another process participating. In particular if a process with the smallest view among non-terminated participates in some iteration, all processes participate in the next iteration (terminated processes are always participating). But if all processes observed in some iteration are participating, then processes return at the end of the round. Hence, a process with a k-set consensus operation terminates at most one round after obtaining the smallest view among non-terminated processes. □ □

Equivalence between the k-test-and-set model and $\mathcal{R}^*_{k-T\&S}$. Both the AS memory and k-set consensus among $k+1$ provides progress to the non-terminated processes with the smallest view infinitely often. Hence, some process will eventually output and terminate as long as they are non-terminated processes. Thus, all processes eventually produces valid task outputs.

We can conclude with the equivalence of the two classes of models. Indeed, this simulation and Algorithm 1 can be used to simulate the affine model $\mathcal{R}^*_{k-T\&S}$ in the k-test-and-set model and reciprocally. Therefore:

Theorem 9. *A task is solvable in the k-test-and-set model if and only if it is solvable in the affine model $\mathcal{R}^*_{k-T\&S}$.*

Thus, we get the following generalization of the asynchronous computability theorem for the k-test-and-set model:

Theorem 10. *Task $T = (\mathcal{I}, \mathcal{O}, \Delta)$ is solvable in the k-test-and-set model if and only if there exist $\ell \in \mathbb{N}$ and a simplicial map $\delta : (\mathcal{R}_{k-T\&S})^\ell(\mathcal{I}) \to \mathcal{O}$ carried by Δ.*

6 Related work and concluding remarks

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 particular 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. [12] 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.

Affine tasks have been defined for the *read-write* models of wait-free [18], t-resilience [25], k-concurrency [11] and, finally, for the general class of fair adversaries [22], encompassing all these models. In this paper, we complement the characterization of [22] with a model in which processes can communicate via k-test-and-set objects, in addition to read-write registers.

This paper proposes 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 [25], 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 $\mathcal{A}_{t-res}$ [25]. 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 5). 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.

References

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

[2] James H Anderson and Mark Moir. Using local-spin k-exclusion algorithms to improve wait-free object implementations. *Distributed Computing*, 11(1):1–20, 1997.

[3] Elizabeth Borowsky and Eli Gafni. Immediate atomic snapshots and fast renaming. In *PODC*, pages 41–51, 1993.

[4] Elizabeth Borowsky and Eli Gafni. A simple algorithmically reasoned characterization of wait-free computation. In *PODC*, pages 189–198, 1997.

[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 *PODC*, pages 311–324, 1990.

[7] Carole Delporte, Hugues Fauconnier, Sergio Rajsbaum, and Michel Raynal. t-resilient immediate snapshot is impossible. In *SIROCCO*, pages 177–191, 2016.

[8] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Andreas Tielmann. The disagreement power of an adversary. *Distrib. Comput.*, 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. *J. ACM*, 32(2):374–382, 1985.

[10] Eli Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony. In *PODC*, pages 143–152, 1998.

[11] Eli Gafni, Yuan He, Petr Kuznetsov, and Thibault Rieutord. Read-Write Memory and k-Set Consensus as an Affine Task. In *OPODIS*, pages 6:1–6:17, 2017.

[12] Eli Gafni, Petr Kuznetsov, and Ciprian Manolescu. A generalized asynchronous computability theorem. In *PODC*, pages 222–231, 2014.

[13] Eli Gafni and Sergio Rajsbaum. Distributed programming with tasks. In *OPODIS*, pages 205–218, 2010.

[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 Trans. Program. Lang. Syst.*, 13(1):123–149, 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. *J. ACM*, 46(2):858–923, 1999.

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

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

[21] Petr Kuznetsov and Thibault Rieutord. Brief Announcement: On Decidability of 2-process Affine Models. In *DISC*, pages 54:1–54:3, 2020.

[22] Petr Kuznetsov, Thibault Rieutord, and Yuan He. An asynchronous computability theorem for fair adversaries. In *PODC*, pages 387–396, 2018.

[23] 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.

[24] Thibault Rieutord. *Combinatorial characterization of asynchronous distributed computability*. PhD thesis, Université Paris Saclay (COmUE), October 2018.

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

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

A Simulating AS Memory Operations in the IIS Model

Simulating the wait-free model in the IIS model is more complicated than solving the IS task. Indeed, we must ensure that persistent operations are simulated on top of a communication-closed abstraction. Therefore, a *slow* process cannot terminate its simulated update operations as long as fast processes might still perform snapshot operations, which should return the last completed update operations of all processes. But since there are no failures in the IIS model, all processes must produce valid outputs to solve a task.

The solution to this issue consists of ensuring that some non-terminated process progresses with its simulation, i.e., completing an unbounded number of AS memory operations. Hence, when executing a task solution, this process will eventually terminate as correct processes must return with a task output. Thus, as long as there are non-terminated processes, one non-terminated process will obtain a task output and terminate its simulation, allowing slower processes also to terminate until all processes get a task output.

Algorithm's description. We present, in Algorithm 4, the simulation proposed in [13] which simplifies the original simulation proposed in [4]. Another simulation was proposed in [5], which ensures that all "fast" processes make progress with the simulation. But as any execution is a run of the wait-free model, guaranteeing that at least one non-terminated process makes progress is sufficient.

Algorithm 4: Simulation of the AS memory in the IIS model for p_i.

```
1  Shared Objects: infinite sequence of IS memories M_1, M_2, ...;
2  Init: SIM_MEM[1,...,n] initialized to ⊥, SIM_counters[1,...,n] initialized to 0;

3  IS_round = 1, SIM_MEM[i] = Initial_State_i, SIM_counters[i] = 1;

4  Do
5  |  IS_View ← M_round.WriteSnapshot(SIM_MEM[i], SIM_counters[i]);
6  |  forall (V, C) ∈ IS_View do
7  |  |  for j = 1,...,n do
8  |  |  |  if C[j] > SIM_counters[j] then
9  |  |  |  |  SIM_MEM[j] ← V[j], SIM_counters[j] ← C[j];
   |
10 |  sum = 0, for j = 1,...,n do sum = sum + SIM_counters[j];
11 |  if (Undecided) ∧ (IS_round = sum) then
12 |  |  SimulateSnapshot(SIM_MEM);
13 |  |  if Decision(SIM_MEM) ≠ ⊥ then
14 |  |  |  SIM_MEM[i] = ⊥, SimulateDecision(Decision(SIM_MEM));
15 |  |  else SIM_MEM[i] ← SIM_MEM, SIM_counters[i] ← SIM_counters[i] + 1 ;
16 |  IS_round + +;
17 While true;
```

The algorithm is relatively simple. Processes maintain in a local array of n variables, *SIM_MEM*, containing the value of the most recent known simulated write operation for each process. Processes also keep an array of n integers *SIM_counters*, associating write operations in *SIM_MEM* with an operation counter. At each round of the IIS model, processes share as

input to the immediate snapshot all known pending or completed update operations along with the matching operations counter. The immediate snapshot outputs are then used to update *SIM_MEM* and *SIM_counters* with the operations associated with the highest counters.

Afterward, suppose the sum of all operations counters is equal to the IIS round number. In that case, the update operation is completed along with the following simulated snapshot operation using the values from the *SIM_MEM* array. If the process is still *undecided*, then it simulates a new update operation. Otherwise, the process stops increasing its write operation counter and use a particular value $\perp$ instead of its ending update operation value.

Sketch of simulation's correctness. This algorithm will not be modified; consequently, only a rough sketch of its correctness is given. Refer to [13] for more details.

The proof of correctness needs to tackle several aspects. The primary auxiliary proof concerns showing that the sum of operations counters made at line 10 is always smaller or equal to the round counter for undecided processes. It comes directly from a sum increasing with rounds, and strictly increasing in rounds where the sum is equal to the iteration counter. As operation counters increase only with new operations completed by undecided processes, the increase of rounds ensures the liveness of the simulation, i.e., if there is an undecided process, at least one undecided process will eventually complete a new operation.

Showing the safety of the simulation is more intricate as it consists in providing a valid order for the simulated operations. It is done by using the order of the simulated snapshot operations and placing simulated update operations just before the first simulated snapshot operation observing it. Then, showing the consistency of simulated operations boils down to showing that the containment and self-inclusion properties are well inherited from the immediate snapshot properties.

Note that the simulation ensures that processes with the smallest view infinitely often if non-terminated, progress with its shared memory simulation. This aspect is primordial in ensuring that the AS simulation can be combined with other operations, here k-test-and-set operations successfully.