k-Immediate Snapshot and x-Set Agreement: How Are They Related?

Carole Delporte[1], Hugues Fauconnier[1], Sergio Rajsbaum[2], and Michel Raynal[3,4](✉)

[1] IRIF, Université Paris Diderot, Paris, France
[2] Instituto de Matemáticas, UNAM, México D.F 04510, Mexico
[3] Univ Rennes IRISA, CNRS, INRIA, Rennes, France
raynal@irisa.fr
[4] Polytechnic University Hong Kong, Kowloon, Hong Kong

Abstract. An immediate snapshot object is a high level communication object, built on top of a read/write distributed system in which all except one processes may crash. This object provides the processes with a single operation, denoted write_snapshot(), which allows the invoking process to write a value and obtain a set of pairs $\langle$process id, value$\rangle$ satisfying some set containment properties, that represent a snapshot of the values written to the object, occurring immediately after the write step.

Considering an n-process model in which up to t processes may crash, this paper introduces first the k-resilient immediate snapshot object, which is a natural generalization of the basic immediate snapshot (which corresponds to the case $k = t = n - 1$). In addition to the set containment properties of the basic immediate snapshot, a k-resilient immediate snapshot object requires that each set returned to a process contains at least $(n - k)$ pairs.

The paper first shows that, for $k, t < n-1$, k-resilient immediate snapshot is impossible in asynchronous read/write systems. Then it investigates a model of computation where the processes communicate with each other by accessing k-immediate snapshot objects, and shows that this model is stronger than the t-crash model. Considering the space of x-set agreement problems (which are impossible to solve in systems such that $x \leq t$), the paper shows then that x-set agreement can be solved in read/write systems enriched with k-immediate snapshot objects for $x = \max(1, t+k-(n-2))$. It also shows that, in these systems, k-resilient immediate snapshot and consensus are equivalent when $1 \leq t < n/2$ and $t \leq k \leq (n-1) - t$. Hence, the paper establishes strong relations linking fundamental distributed computing objects (one related to communication, the other to agreement), which are impossible to solve in pure read/write systems.

Keywords: Asynchronous system · Atomic read/write register · Computability · Distributed algorithm · Immediate snapshot · Impossibility · k-set agreement · Linearizability · Lower/upper bounds · Process crash · Snapshot object · t-resilience

© Springer Nature Switzerland AG 2020
S. Devismes and N. Mittal (Eds.): SSS 2020, LNCS 12514, pp. 97–112, 2020.
https://doi.org/10.1007/978-3-030-64348-5_8

1 Introduction

Context. This article considers the t-crash model consisting of n asynchronous processes, among which any subset of at most t processes may crash, and communicate through a shared memory composed of single writer/multi reader (SWMR) atomic registers. The $(n-1)$-crash model is also called *wait-free* model [12]. We keep the term t-resilience for algorithms. This article focuses on algorithms for distributed tasks in which every non-failed process has to produce an output value (*wait-freedom* progress condition[1]).

A task is defined in terms of (a) possible inputs to the processes, and (b) valid outputs for each assignment of input values (tasks are precisely defined in [6,15,17]). Of special importance is the family of x-*set agreement* tasks [8], one for each integer value of x, $1 \le x \le n$. Set agreement was introduced to show a hierarchy of tasks whose solvability depends on t, the number of processes that may crash. In the x-set agreement task, processes decide at most x different values, out of their input assignments. When $x = 1$, x-set agreement is the celebrated *consensus* (CONS) task. Consensus is impossible even in the presence of a single process crash [19], and $(n-1)$-set agreement is wait-free impossible, namely, in the presence of $n-1$ process crashes [3,17,23], a result proved using algebraic topology. More generally, x-set agreement is solvable if and only if $t < x$, as implied by the simulation in [6]. There are characterizations of the solvability of any given task, in the t-crash model, and in others (for an overview of results see [13]).

Immediate Snapshot Object. The *immediate snapshot* (IS) object was first used in [4,23], and then further investigated as an "object" in [3]. This object is at the heart of the *iterated immediate snapshot* (IIS) model [5,16], which consists of n asynchronous wait-free processes, communicating through IS objects. In an *iterated* model [21], the processes execute a sequence of asynchronous rounds, and each round is provided with exactly one object, which allows the processes to communicate only during this round. In the IIS model, for any $r > 0$, a process accesses the r^{th} immediate snapshot object only when it executes the r-th round, and accesses it only once.

From an abstract point of view, an IS object *IS*, can be seen as an initially empty set, which can then contain up to n pairs (one per process), each made up of a process index and a value. This object provides each process with a single operation denoted write_snapshot(), that it can invoke once. The invocation *IS*.write_snapshot(v) by a process p_i adds the pair $\langle i, v\rangle$ to *IS* and returns a set of pairs belonging to *IS* such that the sets returned to the processes that invoke write_snapshot() satisfy specific inclusion properties. It is important to notice that, in the IIS model, the processes access the sequence of IS objects one after the other, in the same order, and asynchronously. The power of the IIS model with respect to task solvability is the same as the one of the classical read/write

[1] Weaker progress conditions, such as obstruction-freedom [14] and non-blocking [18] have been proposed for $(n-1)$-resilient algorithms.

model, its interest lies in the fact that it provides a higher abstraction layer than the read/write model; a survey including simulations between iterated and classical models can be found in [15].

Contribution of the Paper. This work continues and generalizes the work started in [9] where a preliminary result was presented. Roughly speaking, while [9] considered the case $k = t$, the present article addresses the more general case $k \leq t$.

As previously said, the IS object was designed for the wait-free model (i.e., $t = n-1$). This paper considers it in the context of the t-crash n-process system models where $t < n-1$. To this end it generalizes the IS object by introducing the notion of a *k-immediate snapshot* (k-IS) object. Such an object provides the processes with a single operation denoted $\mathsf{write\_snapshot}_k()$ which, in addition to the properties of an IS object, returns a set including at least $(n-k)$ pairs. Hence, for $k < n-1$, due to the implicit synchronization implied by the constraint on the minimal size of the sets it returns, a k-IS object allows processes to obtain more information from the whole set of processes than a simple IS object (which may return sets containing less than $(n-k)$ pairs).

The obvious question is then the implementability of a k-IS object in the t-crash n-process asynchronous read/write model. The paper shows first that, differently from the basic IS object which can be implemented in the wait-free model, no k-IS object where $k < n-1$, can be implemented in a 1-crash n-process read/write system.

This impossibility result is far from being the first impossibility result in the presence of asynchrony and process crashes, e.g. see the monograph [2]. We already mentioned the impossibility of Consensus (CONS) in the presence of even a single process crash and the impossibility of x-set agreement (x-SA) when $x \leq t$. These agreement objects are at the heart of the theory of fault-tolerant distributed computing. Hence, a second natural question: Are there relations linking the previous "impossible" objects, namely k-IS and x-SA, and if the answer is "yes", under which conditions? The paper provides the following answers to this question[2].

- Let $1 \leq k \leq t < n$. It is possible to implement a k-IS object in a t-crash n-process read/write system enriched with consensus objects.
- Let $1 \leq t < n/2$ and $t \leq k \leq (n-1)-t$. k-IS and Consensus are equivalent in a t-crash n-process read/write system. (A and B are equivalent if A can be implemented in the t-crash n-process read/write system enriched with B, and reciprocally.)

[2] As already indicated, this work was initiated in [9]. Considering k-IS in a system in which up to k processes may crash, this preliminary result showed that, somehow surprisingly, while there is a deterministic $(n-1)$-resilient algorithm implementing an $(n-1)$-IS object in an $(n-1)$-crash read/write system, there is no t-resilient algorithm that implements a t-IS object when $1 \leq t < n-1$.

Table 1. From k-IS to x-SA for $n = 11$ and $x = \mathsf{max}(1, t + k - (n - 2))$

$k \rightarrow$ $t \downarrow$	1 1	2 2	3 3	.. 4	.. 5	.. 6	$n-4$ 7	$n-3$ 8	$n-2$ 9	$n-1$ 10
1	1-SA	1-SA	1-SA	1-SA	1-SA	1-SA	1-SA	1-SA	1-SA	2-SA
2		1-SA	1-SA	1-SA	1-SA	1-SA	1-SA	1-SA	2-SA	3-SA
3			1-SA	1-SA	1-SA	1-SA	1-SA	2-SA	3-SA	4-SA
4				1-SA	1-SA	1-SA	2-SA	3-SA	4-SA	5-SA
$5 < n/2$					1-SA	2-SA	3-SA	4-SA	5-SA	6-SA
$6 \geq n/2$						3-SA	4-SA	5-SA	6-SA	7-SA
$7 = n-4$							5-SA	6-SA	7-SA	8-SA
$8 = n-3$								7-SA	8-SA	9-SA
$9 = n-2$									9-SA	10-SA
$10 = n-1$										11-SA

– Let $(n-1)/2 \leq k \leq n-1$ and $(n-1)-k \leq t \leq k$. It is possible to implement an x-SA object, where $x = t + k - (n-2)$, in a t-crash n-process read/write system enriched with k-IS objects.

An illustration of the results is presented in Table 1, which considers a system of $n = 11$ processes. As an example, the entry $\langle 4, n-4 \rangle$ states that, in the presence of up to $t = 4$ crashes, $(n-4)$-IS allows to solve 2-SA.

Roadmap. The paper is made up of 7 sections. Section 2 presents the basic t-crash n-process asynchronous read/write model, and the definitions of the IS, x-SA, and k-IS objects. Section 3 proves the impossibility for the k-IS object in the previous basic model. The other sections are on the power of k-IS with respect to x-SA. Section 4 shows that x-SA can be built in the t-crash n-process asynchronous read/write model enriched with k-IS objects, for $x = \mathsf{max}(1, t + k - (n-2))$. Section 5 shows that t-IS and CONS are equivalent in the t-crash n-process asynchronous read/write model when $1 \leq t < n/2$. Section 6 shows that CONS is stronger than k-IS when $n/2 \leq t \leq k < n-1$. Finally, Sect. 7 concludes the paper.

2 The Model and the Problems

2.1 Basic Read/Write System Model

Processes. The computing model is composed of a set of $n \geq 3$ sequential processes denoted p_1, ..., p_n. Each process is asynchronous which means that it proceeds at its own speed, which can be arbitrary and remains always unknown to the other processes.

A process may halt prematurely (crash failure), but executes correctly its local algorithm until it possibly crashes. The model parameter t denotes the maximal number of processes that may crash in a run. A process that crashes in a run is said to be *faulty*. Otherwise, it is *correct* or *non-faulty*. Let us notice that, as a faulty process behaves correctly until it crashes, no process knows if it is correct or faulty. Moreover, due to process asynchrony, no process can know if another process crashed or is very slow.

It is assumed that (a) $t < n$ (at least one process does not crash), and (b) any process, until it possibly crashes, executes correctly the algorithm assigned to it. Moreover, each process is assumed to participate in the algorithm.

Communication Layer. The processes cooperate by reading and writing Single-Writer Multi-Reader (SWMR) atomic read/write registers. This means that the shared memory can be seen as a set of variables $A[1..n]$ where, while $A[i]$ can be read by all processes, it can be written only by p_i.

Notation. The previous model is denoted $\mathcal{CARW}_{n,t}[\emptyset]$ (which means "Crash Asynchronous Read/Write with n processes, among which up to t may crash"). A model constrained by a predicate on t (e.g. $t < a$) is denoted $\mathcal{CARW}_{n,t}[t < a]$. $\mathcal{CARW}_{n,t}[t = n-1]$ is a synonym of $\mathcal{CARW}_{n,t}[\emptyset]$, which (as already indicated) is called *wait-free* model. When considering t-crash models, $\mathcal{CARW}_{n,t}[t < a]$ is less constrained than $\mathcal{CARW}_{n,t}[t < a - 1]$. More generally, $\mathcal{CARW}_{n,t}[P, T]$ denotes the system model $\mathcal{CARW}_{n,t}[\emptyset]$ restricted by the predicate P, and enriched with any number of shared objects of the type T (e.g., consensus objects).

Shared objects are denoted with capital letters. The local variables of a process p_i are denoted with lower case letters, sometimes suffixed by the process index i.

2.2 Immediate Snapshot (IS)

The immediate snapshot (IS) object [3] was informally presented in the introduction. Defined in the context of the wait-free model (i.e., $t = n - 1$), it can be seen as a variant of the snapshot object introduced in [1]. While a snapshot object provides the processes with two operations (write() and snapshot()) which can be invoked separately by a process (usually a process invokes write() before snapshot()), a one-shot immediate snapshot object provides the processes with a single operation write_snapshot() (one-shot means that a process may invoke write_snapshot() at most once).

Definition. Let *IS* be an IS object. It is a set, initially empty, that will contain pairs made up of a process index and a value. Let us consider a process p_i that invokes *IS*.write_snapshot(v). This invocation adds the pair $\langle i, v\rangle$ to *IS* (contribution of p_i to *IS*), and returns to p_i a set, called view and denoted $view_i$, such that the sets returned to processes (that return from their invocation of write_snapshot()) collectively satisfy the following properties.

– Termination. The invocation of write_snapshot() by a correct process terminates.

- Self-inclusion. $\forall\, i:\ \langle i, v\rangle \in view_i$.
- Validity. $\forall\, i: (\langle j, v\rangle \in view_i) \Rightarrow p_j$ invoked write_snapshot(v).
- Containment. $\forall\, i, j:\ (view_i \subseteq view_j) \vee (view_j \subseteq view_i)$.
- Immediacy. $\forall\, i, j:\ (\langle i, v\rangle \in view_j) \Rightarrow (view_i \subseteq view_j)$.[3]

Implementations of an IS object in the wait-free model $\mathcal{CARW}_{n,t}[t = n-1]$ are described in [3,22]. While both a one-shot snapshot object and an IS object satisfy the Self-inclusion, Validity and Containment properties, only an IS object satisfies the Immediacy property. This additional property creates an important difference, from which follows that, while a snapshot object is atomic (operations on a snapshot object can be linearized [18]), an IS object is not atomic (its operations cannot always be linearized). However, an IS object is set-linearizable (set-linearizability allows several operations to be linearized at the same point of the time line [7,20]).

*The Iterated Immediate Snapshot (*IIS*) Model.* This model (introduced in [5]) considers $t = n-1$. Its shared memory is composed of a (possibly infinite) sequence of IS objects: $IS[1]$, $IS[2]$, ..., which are accessed sequentially and asynchronously by the processes according to the following round-based pattern executed by each process p_i. The variable r_i is local to p_i; it denotes its current round number.

```
$r_i \leftarrow 0$; $\ell s_i \leftarrow$ initial local state of $p_i$ (including its input, if any);
**repeat forever** % asynchronous IS-based rounds
   $r_i \leftarrow r_i + 1$;
   $view_i \leftarrow IS[r_i]$.write_snapshot$(\ell s_i)$;
   computation of a new local state $\ell s_i$ (which contains $view_i$)
**end repeat**.
```

As indicated in the Introduction, when considering distributed tasks (as formally defined in [6,15,17]), the IIS model and $\mathcal{CARW}_{n,t}[t = n-1]$ have the same computability power [5,11,15].

2.3 x-Set Agreement (x-SA)

x-Set agreement was introduced by S. Chaudhuri [8] to investigate the relation linking the number x of different values that can be decided in an agreement problem, and the maximal number of faulty processes t. It generalizes consensus which corresponds to the instance $x = 1$.

An x-set agreement (x-SA) object is a one-shot object that provides the processes with a single operation denoted propose$_x()$. This operation allows the invoking process p_i to propose a value, which is called *proposed* value, and is passed as an input parameter. It returns a value, called *decided* value. The object is defined by the following set of properties.

[3] An equivalent formulation of the Immediacy property is: $\forall\, i, j:\ \big((\langle i, -\rangle \in view_j) \wedge (\langle j, -\rangle \in view_i)\big) \Rightarrow (view_i = view_j)$.

- Termination. The invocation of $\mathsf{propose}_x()$ by a correct process terminates.
- Validity. A decided value is a proposed value.
- Agreement. No more than x different values are decided.

It is shown in [4,17,23] that $(n-1)$-SA is impossible to implement in $\mathcal{CARW}_{n,t}[t=n-1]$, and in [6] that x-SA is impossible to implement in $\mathcal{CARW}_{n,t}[x \le t]$.

2.4 *k*-Immediate Snapshot

Definition of k-Immediate Snapshot. A k-immediate snapshot (k-IS) object is an immediate snapshot object with the following additional property.

- Output size. The set $view$ obtained by a process is such that $|view| \ge n-k$.

This means that in addition to the Self-inclusion, Validity, Containment, and Immediacy properties, the set returned by a process contains at least $(n-k)$ pairs. The associated operation is denoted $\mathsf{write\_snapshot}_k()$.

k-Immediate Snapshot vs x-Set Agreement. When considering a k-IS object and a x-SA object, we have the following differences.

- On concurrency. An x-SA object is atomic (linearizable), while a k-IS object is not (it is only set-linearizable [7,20]). In other words, k-IS objects "accept" concurrent accesses (this is captured by the Immediacy property), while x-SA objects do not.
- On the values returned. When considering an x-SA object, each process p_i knows that each other process p_j (which returns from its invocation of $\mathsf{propose}_x()$) obtains a single value, but it does not know which one (uncertainty); p_i knows only that at most k values are decided by all processes (certainty).
  When considering a k-IS object, each process p_i knows that each other process p_j (which returns from its invocation of $\mathsf{write\_snapshot}_k()$) obtains a set of pairs $view_j$ that is included in, is equal to, or includes its own set of pairs (certainty due to the containment property), but it does not know the size of $view_j$ (uncertainty).

A Property Associated with k-IS Objects. The next theorem (stated and proved in [9]) characterizes the power of a k-IS object in term of its Output size and Containment properties.

Theorem 1. *Let us consider a k-IS object, and assume that all correct processes invoke $\mathsf{write\_snapshot}_k()$. If the size of the smallest view obtained by a process is ℓ ($\ell \ge n-k$), there is a set S of processes such that $|S| = \ell$ and each process of S obtains the smallest view or crashes during its invocation of $\mathsf{write\_snapshot}_k()$.*

Proof. It follows from the Output size property of the k-IS object that no view contains less than $\ell \ge n-k$ pairs. Let min_view be the smallest view returned by a process; hence $\ell = |min\_view|$.

Let us consider a process p_i such that $(\langle i, -\rangle \in min\_view)$, which returns a view. Due to (a) the Immediacy property (namely $(\langle i, -\rangle \in min\_view) \Rightarrow (view_i \subseteq min\_view))$ and (b) the minimality of min_view, it follows that $view_i = min\_view$. As this is true for each process whose pair participates in min_view, it follows that there is a set S of processes such that $|S| = \ell \geq n-k$, and each of these processes obtains min_view, or crashes during its invocation of $\mathsf{write\_snapshot}_k()$. Due to the Containment property, the others processes crash or obtain views which are a superset of min_view. $\Box_{Theorem\,1}$

An Impossibility Result. The following theorem first stated and proved in [9] establishes an important property of a k-IS object.

Theorem 2. *A* k-IS *object cannot be implemented in* $\mathcal{CARW}_{n,t}[k < t]$.

Proof. To satisfy the output size property, the view obtained by a process p_i must contain pairs from $(n-k)$ different processes. If t processes crash (e.g., initial crashes), a process can obtain at most $(n-t)$ pairs. If $t > k$, we have $n-t < n-k$. It follows that, after it has obtained pairs from $(n-t)$ processes, a process can remain blocked forever waiting for the $(t-k)$ missing pairs. $\Box_{Theorem\,2}$

3 t-Resilience Impossibility of k-Immediate Snapshot

Theorem 3. *It is impossible to implement a* k-IS *object in the model* $\mathcal{CARW}_{n,t}[1 \leq t \leq k < n-1]$.

Proof. The proof considers the case $1 = t \leq k < n-1$ (this constraint explains the model assumption $n \geq 3$, Sect. 2.1). If, for $k \leq n-1$, there is no implementation of a k-IS object in $\mathcal{CARW}_{n,t}[t = 1]$, there is no implementation either for $t \geq 1$. The proof is by contradiction, namely, assuming an implementation of a k-IS object, where $k < n-1$, in $\mathcal{CARW}_{n,t}[t = 1]$, we show that it is possible to solve consensus in $\mathcal{CARW}_{n,t}[t = 1, k\text{-IS}]$. As consensus cannot be solved in $\mathcal{CARW}_{n,t}[t = 1]$, it follows that k-IS cannot be implemented in $\mathcal{CARW}_{n,t}[1 \leq t \leq k]$.

Let us recall the main property of k-IS (captured by Theorem 1). Let ℓ be the size of the smallest view (min_view) returned by a process. There is a set S of ℓ processes such that any process of S returns min_view or crashes, and $\ell \geq n-k$. As $k < n-1$ (theorem assumption), we have $\ell \geq 2$, which means that at least two processes obtain min_view. It follows that, if a process obtains the views returned by the k-IS object to $(n-1)$ processes, one of these views is necessarily min_view. This constitutes Observation O.

Let us now consider Algorithm 1. In addition to a k-IS object denoted IS, the processes access an array $VIEW[1..n]$ of SWMR atomic registers, initialized to $[\bot, \cdots, \bot]$. The aim of $VIEW[i]$ is to store the view obtained by p_i from the k-IS object IS. When it calls $\mathsf{propose}_1(v)$, a process p_i invokes first the k-IS object, in which it deposits the pair $\langle i, v\rangle$ and obtains a view from it (line 1), that it writes in $VIEW[i]$ to make it publicly known (line 2). Then, it waits

```
operation propose_1(v) is
(1)   $view_i \leftarrow IS.\mathsf{write\_snapshot}_k(v)$;
(2)   $VIEW[i] \leftarrow view_i$;
(3)   $\mathsf{wait}(|\{\ j$ such that $VIEW[j] \neq \bot\}| = n - t)$;
(4)   let $view$ be the smallest of the previous $(n - t)$ views;
(5)   return(smallest proposed value in $view$)
end operation.
```

Algorithm 1: Solving consensus in $\mathcal{CARW}_{n,t}[t = 1, k\text{-IS}]$ (code for p_i)

until it sees the views of at least $(n - 1)$ processes (line 3). Finally, p_i extracts from these views the one with the smallest cardinality (line 4), and returns the smallest value contained in this smallest view (line 5).

We show that this reduction algorithm solves consensus in $\mathcal{CARW}_{n,t}[t = 1, k\text{-IS}]$. As at least $(n - 1)$ processes do not crash, and write in their entry of the array $VIEW[1..n]$, no correct process can block forever at line 2, proving the Termination property of consensus.

As $\ell \geq n - k \geq 2$, it follows from Observation O that at least one of the views obtained by a process at line 3 is necessarily min_view. It follows that each process that executes line 3 obtains min_view and returns its smallest value at line 4), proving the Agreement property of consensus.

The consensus Validity property follows directly from k-IS Validity property, and the observation that any set $view$ contains only proposed values line 4). $\Box_{Theorem\ 3}$

Remark. When considering Algorithm 1, let us observe that, as $n - k \leq n - t$, the array $VIEW[1..n]$ can be replaced by a k-immediate snapshot object $IS2$. We obtain then the following algorithm.

```
operation propose_1(v) is
        $view1_i \leftarrow IS.\mathsf{write\_snapshot}_k(v)$;
        $view2_i \leftarrow IS2.\mathsf{write\_snapshot}_k(view1_i)$;
        let $view$ be the smallest view in $view2_i$;
        return(smallest proposed value in $view$)
end operation.
```

4 From k-Immediate Snapshot to x-Set Agreement

This section proves the content of Table 1, namely x-SA can be implemented in the system model $\mathcal{CARW}_{n,t}[t \leq k < n - 1]$, for $x = \max(1, t + k - (n - 2))$. Interestingly, the algorithm providing such an implementation is Algorithm 1, whose operation name is now $\mathsf{propose}_x()$ (instead of $\mathsf{propose}_1(v)$).

Theorem 4. *Let* $x = \max(1, k + t - (n - 2))$. *Algorithm 1 implements an* x-SA *object in the system model* $\mathcal{CARW}_{n,t}[1 \leq t \leq k < n - 1, k\text{-IS}]$.

Proof. The x-SA Termination follows directly from the Termination property of the underlying k-IS object IS, the fact that there are at least $(n-t)$ correct processes, and the assumption that all correct processes invoke $\mathsf{propose}_x()$. The x-SA Validity property follows directly from the Validity property of the IS.

As far as the x-SA Agreement property is concerned, we have the following. Due to Theorem 1, a set of $\ell \geq n-k$ processes obtain the smallest possible view min_view, which is such that $|min\_view| = \ell \geq n-k$. It follows that, at most k processes obtain a view different from min_view. In the worst case, these k views are different. Consequently, there are at most $k+1$ different views, namely min_view, $V(1)$, ..., $V(k)$, and due to their Containment property, we have $min\_view \subset V(1) \subset \cdots \subset V(k)$. The rest of the proof is a case analysis according to the value of $(n-t)$ with respect to k.

- $n-t > k$. In this case, a process obtains views from $(n-t)$ processes (line 3), and in the first case it obtains the views $V(1)$, ..., $V(k)$. But as $n-t>k$ it also obtains min_view from at least one process. It follows that, all processes see min_view, and consequently decide the same value at line 5. Hence, $(n-t>k) \Rightarrow (x=1)$.
- $n-t=k$. In this case, it is possible that some processes do not obtain min_view at line 3. But, if this occurs, they necessarily obtain the views from the $n-t=k$ processes that deposited $V(1)$, ..., $V(k)$ in $VIEW[1..n]$. Hence, all these processes obtains $V(1)$ at line 3, and decide consequently the same value from $V(1)$. As the decided values are decided from the views min_view and $V(1)$, we have $(n-t=k) \Rightarrow (x=2)$.
- $n-t=k-1$. In this case, it is possible that, at line 3, some processes obtain not only min_view, but also $V(1)$ and decide the smallest value of $V(2)$. As the decided values are then decided from the views min_view, $V(1)$, and $V(2)$, we have $(n-t=k-1) \Rightarrow (x=3)$.
- Applying the same reasoning to the general case $n-t=k-c$, we obtain $(n-t=k-c) \Rightarrow (x=2+c)$.

Abstracting the previous case analysis, we obtain $x=1$ (consensus) for $n-t>k$, and $x=k+t-(n-2)$, i.e., when $n-t=k-x+2$, from which follows that $x=\mathsf{max}(1, k+t-(n-2))$, which completes the proof of the theorem. $\square_{Theorem\,4}$

The next corollary is a re-statement of Theorem 4 for $x=1$.

Corollary 1. *Algorithm 1 implements a* CONS *object in the system model* $\mathcal{CARW}_{n,t}[1 \leq t < n/2, t \leq k \leq (n-1)-t, k\text{-IS}]$.

5 An Equivalence Between k-Immediate Snapshot and Consensus

This section shows first that consensus is strong enough to implement a k-IS object when $t \leq k$. Combining this result with the fact consensus can be implemented from a k-IS object in $\mathcal{CARW}_{n,t}[1 \leq t < n/2, t \leq k \leq (n-1)-t]$ (Corollary 1), we obtain that consensus and k-IS are equivalent in $\mathcal{CARW}_{n,t}[1 \leq t < n/2, t \leq k \leq (n-1)-t]$.

5.1 From CONS to k-IS in $\mathcal{CARW}_{n,t}[t \le k \le n-1]$

Algorithm 2 describes a reduction of k-IS to consensus in $\mathcal{CARW}_{n,t}[0 < t \le k \le n-1]$. This algorithm uses three shared data structures. The first is an array $REG[1..n]$ of SWMR atomic registers (where $REG[i]$ is associated with p_i), the second is a consensus object denoted CS, and the third is an immediate snapshot object denoted IS (let us recall that such an object can be implemented in $\mathcal{CARW}_{n,t}[t \le n-1]$).

```
operation write_snapshot_k($v_i$) is
(1)  $REG[i] \leftarrow v_i$;
(2)  wait $\big(|j$ such that $REG[j] \neq \perp\}| \geq n-k\big)$;
(3)  $aux_i \leftarrow \{\langle j, REG[j]\rangle$ such that $REG[j] \neq \perp\}$;
(4)  $view_i \leftarrow CS.\mathsf{propose}_1(aux_i)$;
(5)  if $(\langle i, v_i\rangle \in view_i)$
(6)     then return($view_i$)
(7)     else $aux_i \leftarrow IS.\mathsf{write\_snapshot}(v_i)$;
(8)          $view_i \leftarrow view_i \cup aux_i$;
(9)          return($view_i$)
(10) end if
end operation.
```

Algorithm 2: Building k-IS in $\mathcal{CARW}_{n,t}[0 < t \le k \le n-1, \text{CONS}]$ (code for p_i)

The behavior of a process p_i can be decomposed in three parts.

- When it invokes $\mathsf{write\_snapshot}_k(v_i)$, p_i first deposits its value v_i in $REG[i]$, in order all processes to know it, and waits until at least $(n-k)$ processes have deposited their input value in $REG[1..n]$ (lines 1–2).
- Then p_i proposes to the underlying consensus object CS, the set of all the pairs $\langle j, REG[j]\rangle$ such that $REG[j] \neq \perp$ (lines 3–4). Let us notice that this set contains at least $(n-k)$ pairs. Hence, the consensus object returns to p_i a view $view_i$, which contains at least $(n-k)$ pairs.
- Finally, p_i returns a view (of at least $(n-k)$ pairs).
  - If $view_i$ contains its own pair $\langle i, v_i\rangle$, p_i returns $view_i$ (line 6).
  - If $view_i$ does not contain $\langle i, v_i\rangle$, p_i proposes v_i to the underlying immediate snapshot object from which it obtains a set of pairs aux_i (line 7). Let us notice that, due to the properties of the immediate snapshot object IS, aux_i contains the pair $\langle i, v_i\rangle$. Process p_i then adds aux_i to $view_i$ (line 8) and returns it (line 9).

Theorem 5. *Algorithm 2 implements a k-IS object in the system model $\mathcal{CARW}_{n,t}[0 < t \le k \le n-1, \text{CONS}]$.*

Proof. Proof of k-IS Self-inclusion. If p_i returns at line 6, self-inclusion follows directly from the predicate of line 5. If this predicate is not satisfied, p_i invokes the underlying immediate snapshot object IS with the value v_i it initially proposed (line 7). It then follows from the self-inclusion property of IS that aux_i contains $\langle i, v_i \rangle$, and due to line 8, the set $view_i$ that is returned at line 9 contains $\langle i, v_i \rangle$.

Proof of k-IS Validity. This property follows from (a) the fact that a process p_i assigns to $REG[i]$ the value it wants to deposit in the k-IS object, (b) this atomic variable is written at most once (line 1), and (c) the predicate $REG[j] \neq \bot$ is used at line 3 to extract values from $REG[1..n]$.

The Output size property follows from (a) the predicate of line 2, which ensures that the set $view_i$ obtained at line 4 from the underlying consensus object contains at least $n - t \geq n - k$ pairs, and the fact that a set $view_i$ cannot decrease (line 3).

Proof of k-IS Containment. Let P6 (resp., P9) be the set of processes that terminate at line 6 (resp., 9). Let $view$ be the set of pairs decided by the underlying consensus object CS (line 4). Hence, all the processes in P6 return $view$. Due to line 8, the set $view_i$ returned by a process that terminates at line 9 includes $view$. It follows that $\forall\ p_j \in$ P6, $p_i \in$ P9, we have $view_j = view \subset view_i$.

Let us now consider two processes p_i and p_j belonging to P9. It then follows from the IS Containment property of the underlying IS object, that we have $aux_i \subseteq aux_j$ or $aux_j \subseteq aux_i$ (where the value of aux_i and aux_j are the ones at line 7). Consequently, at line 8 we have $view_i \subseteq view_j$ or $view_j \subseteq view_i$, which completes the proof of the k-IS Containment property.

Proof of k-IS Immediacy. Let p_i and p_j be two processes that return $view_i$ and $view_j$, respectively, such that $\langle i, v \rangle \in view_j$. We have to show that $view_i \subseteq view_j$. Let us considering the sets P6 and P9 defined above. There are three cases.

- Both p_i and p_j belong to P6. In this case, due to line 4, we have $view_i = view_j$.
- p_i belongs to P6, while p_j belong to P9. In this case, due to line 8, we have $view_i \subset view_j$.
- Both p_i and p_j belong to P9. In this case, due to the IS Immediacy property of IS we have (at line 8) $\langle i, - \rangle \in aux_j \Rightarrow aux_i \subseteq aux_j$ (and $\langle j, - \rangle \in aux_i \Rightarrow aux_j \subseteq aux_i$). Let $view$ the set of pairs returned by the consensus object line 4. As, due to line 9, we have $view_i \leftarrow view \cup aux_i$ and $view_j \leftarrow view \cup aux_j$, the k-IS Immediacy property follows.

Proof of k-IS Termination. Let p be the number of processes that deposit a value in REG. As $t \leq k$, we have $n - k \leq n - t \leq p \leq n$. It follows that no correct process can wait forever at line 2. The fact that no correct process blocks forever at line 4 and line 7 follows from the termination property of the underlying consensus and immediate snapshot objects. $\Box_{Theorem\, 5}$

5.2 When Consensus and k-IS Are Equivalent

Let us consider the right triangular matrix defined by the entries are marked "x-SA" in Table 1. Theorem 5 states that it is possible to implement k-IS from

CONS for any entry (t, k) belonging to this triangular matrix. Combined with Corollary 1, we obtain the following theorem.

Theorem 6. CONS *objects and and* k-IS *objects are equivalent in the system model* $\mathcal{CARW}_{n,t}[0 < t < n/2, t \le k \le (n-1) - t]$.

6 When Consensus Is Stronger Than k-Immediate Snapshot

Section 4 investigated the power of k-IS to implement x-SA objects, namely x-SA can be implemented in $\mathcal{CARW}_{n,t}[1 \le t \le k < n-1, k\text{-IS}]$ where $x = \mathsf{max}(1, t + k - (n-2))$, see Theorem 4. As we have seen, considering the other direction, Sect. 5 has shown that k-IS can be implemented in $\mathcal{CARW}_{n,t}[1 \le t \le k < n-1, \text{CONS}]$ (Theorem 5). The combination of these results showed that Consensus and k-IS are equivalent in $\mathcal{CARW}_{n,t}[0 < t = k < n/2]$ (Theorem 6).

This section shows an upper bound on the power of k-IS to implement x-SA objects, namely, k-IS objects are not powerful enough to implement consensus in the system model $\mathcal{CARW}_{n,t}[n/2 \le t \le k < n-1]$.

Preliminary: A Simple Lemma. Let us remark that, as immediate snapshot objects, k-immediate snapshot objects are not linearizable. As a k-IS object *IS* contains values from at least $(n-k)$ processes, at least $(n-k)$ processes must have invoked the operation $IS.\mathsf{write\_snapshot}_k()$ for any invocation of $\mathsf{write\_snapshot}_k()$ to be able to terminate. It follows that there is a time τ at which $n-k$ processes have invoked $IS.\mathsf{write\_snapshot}_k()$ and have not yet returned. We then say that these $(n-k)$ processes are "*inside IS*". Hence the following lemma.

Lemma 1. *If an invocation of* $\mathsf{write\_snapshot}_k()$ *on a k-immediate snapshot object IS terminates, there is a time τ at which at least $(n-k)$ processes are* inside *IS.*

Theorem 7. *There is no algorithm implementing a* CONS *object in the system model* $\mathcal{CARW}_{n,t}[n/2 \le t \le k < n-1, k\text{-IS}]$.

Due to page limitation, the reader will find the proof of this theorem in [10].

7 Conclusion

The aim and content of the paper. The paper has first introduced the notion of a k-immediate snapshot (k-IS) object, which generalizes the notion of immediate snapshot (IS) objects to t-crash n-process systems (the IS object corresponds to the case $k = t = n-1$). It has then shown that k-IS objects cannot be implemented in asynchronous read/write systems for $k < n-1$.

The paper considered then the respective power of k-IS objects and x-set agreement objects (x-SA) in t-crash-prone systems. As both these families of objects are impossible to implement in read/write systems for $t, k < n-1$ or $x \leq t$, respectively, the paper strove to establish which of k-IS and x-SA objects are the most "impossible to solve". The main results are the following where the zones A, B, C, D, refer to Fig. 1.

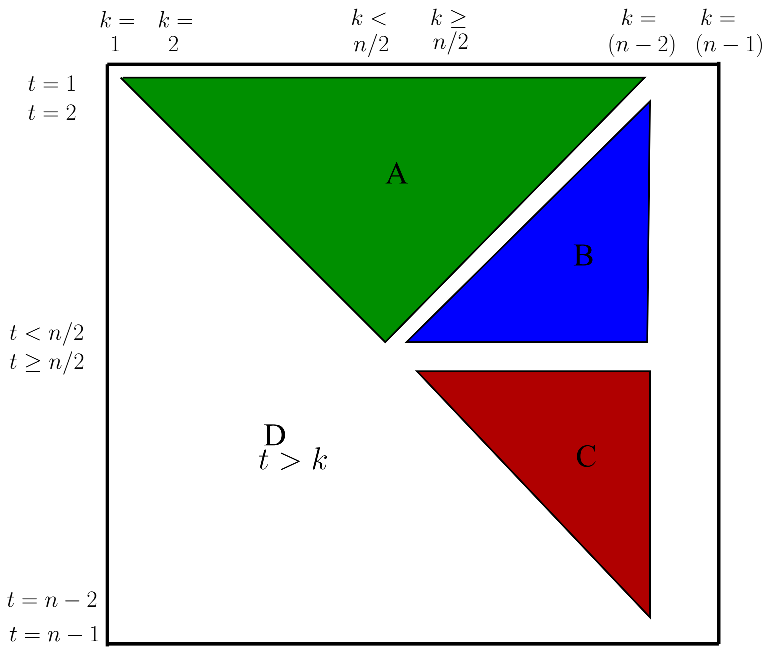

Fig. 1. Summarizing the results

- Even if we have CONS objects, it is not possible to implement k-IS objects in a t-crash system where $t > k$ (Zone D).
- It is possible to implement x-SA objects, where $x = \mathsf{max}(1, t + k - (n-2))$, from k-IS objects in systems where $1 \leq t \leq k < n-1$ (Zone A + B + C).
- It is possible to implement k-IS objects from 1-SA objects (CONS) in read/write systems where $1 \leq t \leq k \leq n-1$ (Zone A + B + C).
- 1-SA objects (CONS) and k-IS objects are equivalent in read/write systems where $1 \leq t < n/2$ and $t \leq k \leq (n-1) - t$ (Zone A).
- It is not possible to implement 1-SA (consensus) from k-IS objects in read/write systems when $n/2 \leq t \leq k < n-1$ (Zone C).

Stated in a more operational way, these results exhibit the price of the synchronization hidden in k-IS object (which requires that the view returned to a process contains at least $(n-k)$ pairs, (where a pair is made up of a value plus the id of the process that deposited it in the k-IS object).

More generally, the previous results establish a computability map relating important problems, which are impossible to solve in pure read/write systems.

Open Problems. The following problems remain to be solved to obtain a finer relation linking k-IS and x-SA, when $t > 1$.

- Direction "from k-IS to x-SA". Is it possible to implement x-SA objects, with $1 \leq x < t+k-(n-2)$ in t-crash n-process systems enriched with k-IS objects (Zone B)? We conjecture that the answer to this question is *no*.
- Direction "from x-SA to k-IS". Given an x-SA object, which k-IS objects can be implemented from it? More generally, is there a "k-IS-like" communication object such that x-SA and this "k-IS-like" object are computationally equivalent (by "k-IS-like" we mean an object possibly weaker than a k-IS object)?

Acknowledgments. The authors want to thank the referees for their constructive comments. This work has been partially supported by the French ANR project DESCARTES (16-CE40-0023-03) devoted to layered and modular structures in distributed computing, and the UNAM-PAPIIT projects IN107714, IN106520.

References

1. Afek, Y., Attiya, H., Dolev, D., Gafni, E., Merritt, M., Shavit, N.: Atomic snapshots of shared memory. J. ACM **40**(4), 873–890 (1993)
2. Attiya, H., Ellen, F.: Impossibility Results for Distributed Computing. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, San Rafael (2014). 162 p
3. Borowsky, E., Gafni, E.: Immediate atomic snapshots and fast renaming. In: Proceedings of the 12th ACM Symposium on Principles of Distributed Computing (PODC 1993), pp. 41–50. ACM Press (1993)
4. Borowsky, E., Gafni, E.: Generalized FLP impossibility results for t-resilient asynchronous computations. In: 25th ACM Symposium on Theory of Computing, pp. 91–100. ACM Press (1993)
5. Borowsky, E., Gafni, E.: A simple algorithmically reasoned characterization of wait-free computations. In: Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC 1997), pp. 189–198. ACM Press (D1997)
6. Borowsky, E., Gafni, E., Lynch, N., Rajsbaum, S.: The BG distributed simulation algorithm. Distrib. Comput. **14**, 127–146 (2001). https://doi.org/10.1007/PL00008933
7. Castañeda, A., Rajsbaum, S., Raynal, M.: Unifying concurrent objects and distributed tasks: interval-linearizability. J. ACM **65**(6), 42 (2018). Article 45
8. Chaudhuri, S.: More choices allow more faults: set consensus problems in totally asynchronous systems. Inf. Comput. **105**(1), 132–158 (1993)
9. Delporte, C., Fauconnier, H., Rajsbaum, S., Raynal, M.: t-resilient immediate snapshot is impossible. In: Suomela, J. (ed.) SIROCCO 2016. LNCS, vol. 9988, pp. 177–191. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48314-6_12
10. Delporte, C., Fauconnier, H., Rajsbaum, S., Raynal, M.: t-resilient k-immediate snapshot and its relation with agreement problems. Technical report, ArXiv:2010.00096, 15 p. (2020)
11. Gafni, E., Rajsbaum, S.: Distributed programming with tasks. In: Lu, C., Masuzawa, T., Mosbah, M. (eds.) OPODIS 2010. LNCS, vol. 6490, pp. 205–218. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17653-1_17

12. Herlihy, M.P.: Wait-free synchronization. ACM Trans. Program. Lang. Syst. **13**(1), 124–149 (1991)
13. Herlihy, M.P., Kozlov, D., Rajsbaum, S.: Distributed Computing Through Combinatorial Topology. Morgan Kaufmann/Elsevier, Amsterdam (2014). 336 p. ISBN 9780124045781
14. Herlihy, M.P., Luchangco, V., Moir, M.: Obstruction-free synchronization: double-ended queues as an example. In: Proceedings of the 23th International IEEE Conference on Distributed Computing Systems (ICDCS 2003), pp. 522–529. IEEE Press (2003)
15. Herlihy, M., Rajsbaum, S., Raynal, M.: Power and limits of distributed computing shared memory models. Theoret. Comput. Sci. **509**, 3–24 (2013)
16. Herlihy, M.P., Shavit, N.: A simple constructive computability theorem for wait-free computation. In: 26th ACM Symposium on Theory of Computing, pp. 243–252. ACM Press (1994)
17. Herlihy, M.P., Shavit, N.: The topological structure of asynchronous computability. J. ACM **46**(6), 858–923 (1999)
18. Herlihy, M.P., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. **12**(3), 463–492 (1990)
19. Loui, M., Abu-Amara, H.: Memory requirements for agreement among unreliable asynchronous processes. Adv. Comput. Res. **4**, 163–183 (1987)
20. Neiger, G.: Set-linearizability. In: Brief Announcement in Proceedings of the 13th ACM Symposium on Principles of Distributed Computing (PODC 1994), p. 396. ACM Press (1994)
21. Rajsbaum, S.: Iterated shared memory models. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 407–416. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12200-2_36
22. Raynal, M.: Concurrent Programming: Algorithms, Principles and Foundations. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-32027-9. 515 p. ISBN 978-3-642-32026-2
23. Saks, M., Zaharoglou, F.: Wait-free k-set agreement is impossible: the topology of public knowledge. SIAM J. Comput. **29**(5), 1449–1483 (2000)