Distributed computability: Relating k-immediate snapshot and x-set agreement

Carole Delporte, Hugues Fauconnier, Sergio Rajsbaum, Michel Raynal

Version of Record: https://www.sciencedirect.com/science/article/pii/S0890540121001450
Manuscript_d138e785eee794dd041522ab33df8676

Distributed Computability: Relating k-Immediate Snapshot and x-Set Agreement

Carole Delporte[†], Hugues Fauconnier[†], Sergio Rajsbaum[◦], Michel Raynal[⋆,‡]

[†] IRIF, Université Paris Diderot, Paris, France
[◦] Instituto de Matemáticas, UNAM, Mexico City, Mexico
[⋆] Univ Rennes IRISA, CNRS, INRIA, France
[‡] Polytechnic University Hong Kong

Abstract

Consider a system of n asynchronous processes out of which t may crash, communicating using read/write shared memory operations. An immediate snapshot object is an important synchronization abstraction that can be implemented in this system, even if all processes except one may crash, $t = n - 1$. It provides processes with a single operation, to write a value and obtain a set of pairs $\langle$process id, value$\rangle$, that represent a snapshot of the values written to the object, occurring immediately after the write step.

This paper introduces a generalization of the immediate snapshot object denoted k-*immediate snapshot,* requiring that the snapshot returned contains at least $(n-k)$ pairs; thus, the case $k = n-1$ corresponds to the original immediate snapshot object, which requires that the snapshot returned contains at least one pair (the $\langle$process id, value$\rangle$ pair, that corresponds to the process id itself that invoked the operation).

The paper first shows that k-immediate snapshot is impossible to implement in an asynchronous read/write system, even if $k = n - 2$ and $t = 1$. Then, the paper considers x-set agreement, another object stronger than the classical read/write t-crash read/write model (when $x \leq t$), and studies the relation with the k-immediate snapshot object, establishing strong relations linking these two fundamental distributed computing abstractions. The paper shows that x-set agreement can be solved in read/write systems enriched with k-immediate snapshot objects for $x \geq \mathsf{max}(1, t + k - (n - 2))$. It also shows that, in these systems, k-immediate snapshot and consensus are equivalent when $1 \leq t < n/2$ and $t \leq k \leq (n - 1) - t$.

Keywords: Asynchronous system, Atomic read/write register, Computability, Distributed algorithm, Immediate snapshot, Impossibility, k-Set Agreement, Process crash, Snapshot object, t-Resilient synchronization.

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. An object may be implemented on this model, to provide an abstraction of higher level, easier to use than SWMR registers. Alternatively, the model may be enriched with stronger objects, that cannot be implemented on top of SWMR registers. An object is defined in terms of (a) inputs to the processes, and (b) valid outputs for each assignment of input values (formal definitions can be found in [6, 15, 17]). We are interested in comparing the power of

different objects. In particular, A and B are equivalent if A can be implemented in the t-crash n-process read/write system enriched with B, and reciprocally.

Of special importance is the family of *x-set agreement* objects [8], one for each integer value of x, $1 \leq x \leq n$, which was introduced to show a hierarchy of objects whose solvability depends on t, the number of processes that may crash. In the x-set agreement object, processes decide at most x different values, out of their input assignments. When $x = 1$, x-set agreement is the celebrated *consensus* (CONS) object, known to be impossible to implement, even in the presence of a single process crash [20]. The x-set agreement object, $x = n - 1$, is impossible in the presence of $t = n - 1$ process crashes [3, 17, 24], a result proved using algebraic topology. More generally, x-set agreement can be implemented if and only if $t < x$, as implied by the simulation in [6]. There are characterizations of the solvability of any given object, in the t-crash model, and in others (for an overview of results see [14]).

Immediate snapshot object. The *immediate snapshot* (IS) object introduced in [4, 24] turned out to be very useful in the theory of distributed computability. 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, illustrated in Figure 1, roughly, each process gets back its own pair, and the sets returned to different processes can be ordered by containment together with an immediacy property; see Section 2.2. In the figure, processes p_1, p_2, p_3 invoke each one IS.write_snapshot(p_i), with its own id as input parameter to the operation. All possible executions can be represented by the triangles depicted in Figure 1. As an example, in the triangle σ_1, process p_1 gets back $\{\langle 1, 1\rangle\}$, process p_2 gets back $\{\langle 1, 1\rangle, \langle 2, 2\rangle\}$, and process p_1 gets back $\{\langle 1, 1\rangle, \langle 2, 2\rangle\ \langle 3, 3\rangle\}$. As another example, the execution depicted by the triangle σ_2, all three processes p_i get back $\{\langle 1, 1\rangle, \langle 2, 2\rangle\ \langle 3, 3\rangle\}$, while in the execution (triangle) σ_3 process p_2 gets the same value, while both the processes p_1 and p_3 get back $\{\langle 1, 1\rangle, \langle 3, 3\rangle\}$.

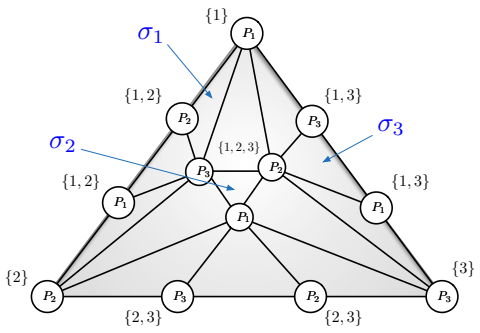

Figure 1: All executions of an immediate snapshot object invoked by p_1, p_2, p_3, where p_i invokes it with input i, correspond to 13 triangles. A vertex contains the name of the process inside, and the value of the pairs it gets back from the invocation outside the vertex.

The IS object is at the heart of the *iterated immediate snapshot* (IIS) model [5, 16, 22], which consists of n asynchronous crash-prone processes, communicating through IS objects only. The processes execute a sequence of asynchronous rounds, each round being provided with exactly one IS object, which allows the processes to communicate only during this round. In the IIS model, for any $r > 0$, a process accesses (only once) the r^{th} immediate snapshot object when it executes the r-th round. Figure 2 shows that all executions of 2 rounds of the IIS model are obtained by recursively replacing each triangle of the 1-round IIS model in Figure 1 by the figure itself.

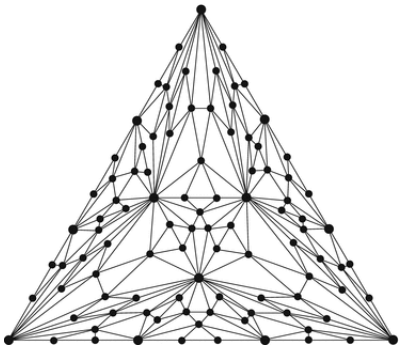

Figure 2: All executions of the 2-round IIS model for three processes..

Motivation. The IS object is very useful when studying the read/write model in where up to $t = n - 1$ processes may crash, because it can be implemented in this model, and furthermore, because the IIS model and the read/write model where $t = n - 1$, have the same computational power to solve tasks [5, 12]. Thus, instead of analyzing the read/write model directly, one can consider the more structured IIS model.

What would be a version of the IS object corresponding to the general t-resilient case, $t \leq n - 1$? We consider the natural candidate, a version of the immediate snapshot object that always returns to an invocation at least $n - t$ pairs. Intuitively, when p_i invokes $IS.\mathsf{write\_snapshot}(v)$, it could write v to its entry of a shared read/write array, and repeatedly read the array until it sees that $n - t$ values have been written, perhaps then take an immediate snapshot of the array. Namely, we are interested in the subset of executions obtained by removing those where a process sees less than $n - t$ pairs, depicted in Figure 3 (triangles touching the corner vertices of the subdivision), for $n = 3, t = 1$.

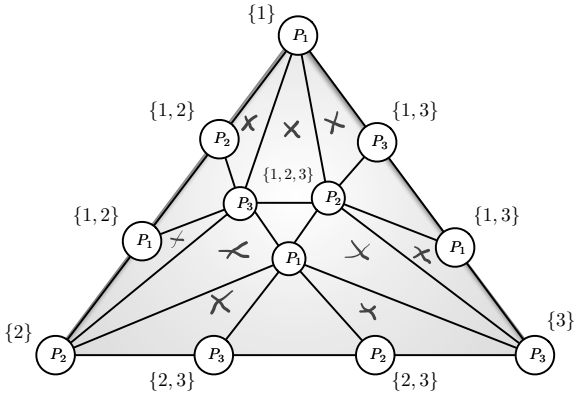

Figure 3: Executions marked with a cross are removed from the 1-round IIS model for three processes executions to obtain a 1-IS object, corresponding to those where a process gets back only one pair, $n = 3, t = 1$.

Remarkably, a similar idea was used in [25], but in the case of two rounds of the IIS model. A *delayed snapshot* object whose executions correspond to a subset of the executions of the 2-round IIS model plays a fundamental role in their proof of a t-resilient task computability theorem. Roughly, executions where a process sees less than $n - t$ pairs in both immediate snapshot objects are removed, as depicted in Figure 4 (again, triangles touching the corner vertices of the subdivision).

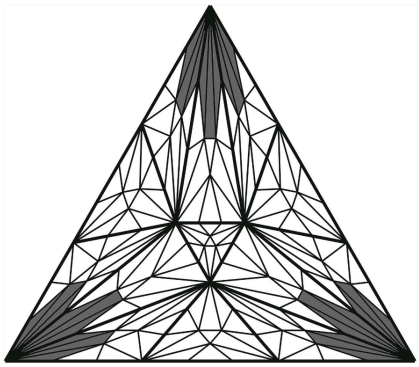

Figure 4: A delayed snapshot object, obtained by removing an execution from the 2-round IIS model for three processes, for $n = 3, t = 1$.

Contribution of the paper. The present paper combines and extends results reported in two conferences ([9] and [10]). The research project started in [9], that, roughly speaking, considered the case $k = t$, and then continued in [10] that studied the more general case $t \leq k$.

As previously mentioned, the IS object was designed for the read/write model in which up to $t = n - 1$ processes may crash. The present paper considers the t-crash n-process model, where $t < n - 1$. It is assumed that all the processes that do not crash participate in the algorithms implementing the operations of the high level objects[1]. 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 previously defined read/write model. The paper shows first that, differently from the basic IS object which can be implemented when $t = n - 1$, no k-IS object where $k < n - 1$, can be implemented in a 1-crash n-process read/write system. This result provides an explanation of why two rounds of IS objects had to be used in [25][2].

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? Notice that the 2-SA object for 3 processes can be illustrated as in Figure 5, with the center triangle removed, which is where processes get back 3 different values. Intuitively, the dual of the 1-IS object, where the triangles incident to the corners, instead of the center, are removed. The paper provides the following answers to this question[3].

[1] This assumption, usual in message-passing systems, is also used in some shared memory models (e.g., [26]). It is related to the synchronization needed to ensure the size requirement on the set of pairs returned by k-IS objects.

[2] And indeed corresponds to the fact that although both objects, our k-IS and the delayed snapshot have the same connectivity, they do not have the same link connectivity.

[3] As already indicated, this work was initiated in [9], which considered k-IS in a system in which up to k processes may crash. This preliminary result showed that, 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 $1 \leq t < n-1$.

An IS object is a snapshot object (a) whose operations $\mathsf{write}()$ and $\mathsf{snapshot}()$ are glued together in a single operation

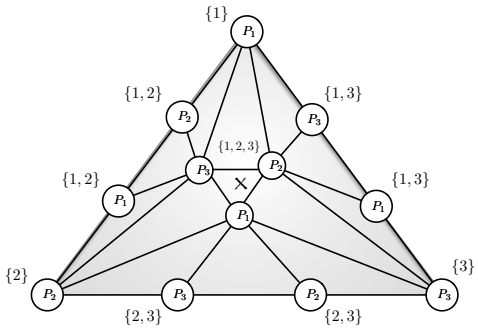

Figure 5: Executions marked with a cross are removed to solve 2-set agreement.

- Let $1 \leq t \leq k < n$. 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.
- 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 $x \geq t + k - (n-2)$, 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. The empty slots in the left/bottom triangle of the table can be filled in with n-SA (11-SA). It is impossible to achieve any k-SA, so we can only achieve objects that are achievable in shared memory.

$k \rightarrow$ $t \downarrow$	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

Table 1: From k-IS to x-SA for $n = 11$ and $x \geq \max(1, t + k - (n-2))$

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. The other sections are on the power of k-IS with respect to x-SA. Section 3 shows that x-SA can be built in the t-crash n-process asynchronous read/write model enriched with k-IS objects, for $x \geq t+k-(n-2)$. Section 4 proves the impossibility for the k-IS object in the previous basic model. Section 5 shows that t-IS and

write_snapshot(), and (b) satisfying an additional property linking the sets of pairs returned by concurrent invocations (called *Immediacy* property, Section 2.2). Then, as already indicated, a t-IS object is an IS object such that the sets returned by write_snapshot() contain at least $(n-t)$ pairs. The same size property on the sets returned by a snapshot object can be trivially implemented in a t-crash n-process model. Let us call t-snapshot such a constrained snapshot object. Hence, while a t-snapshot object can be implemented in the t-crash n-process model, a t-IS object cannot when $0 < t < n-1$.

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, Section 7 concludes the paper.

2 Model and Objects

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.

t-Termination. An algorithm (implementing an object operation) is *t-terminating* (or satisfies the *t-termination* liveness property), if it always terminates despite the crash of at most t processes. An execution is *t-terminating*, if each operation invoked by a correct process is *t-terminating*. The $(n-1)$-termination property is nothing else than the wait-freedom progress condition [13].

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.

Process participation. A process *participates* as soon as it (invokes an operation that) accesses the shared memory. A process that does not participate is considered as a process that initially crashed. It is not known which are these processes[4]. This requirement is needed to ensure the liveness property of the k-immediate snapshot object, which is both an agreement object and a synchronization object.

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 $\mathcal{CARW}_{n,t}[t < a]$. 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 where $t = n-1$, 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

[4] A similar assumption is done in Section 4 of [11], where is presented an n-process consensus algorithm that assumes that a majority of processes are correct and participate in the algorithm, while all the other processes have initially crashed.

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. $\forall\, i, j :\ (\langle i, v\rangle \in view_j) \Rightarrow (view_i \subseteq view_j)$.[5]

Implementations of an IS object in the model $\mathcal{CARW}_{n,t}[t = n-1]$ are described in [3, 23]. 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, 21]).

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 ℓs_i (which contains $view_i$)
end repeat.

As indicated in the Introduction, when considering distributed objects (as formally defined in [6, 15, 17]), the IIS model and $\mathcal{CARW}_{n,t}[t = n-1]$ have the same computability power [5, 12, 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.

[5] 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, 24] 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 $\mathcal{CARW}_{n,t}[x \leq t]$.

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 $|view| \geq n - k$.

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 $\mathsf{write\_snapshot}_k()$. It is important to notice that, for $k > 1$, $\mathsf{write\_snapshot}_k()$ involves an implicit synchronization: $(n - k)$ processes must have deposited a pair in the k-IS object for $\mathsf{write\_snapshot}_k()$ to be able to terminate.

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, 21]). 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 x 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 ($\ell \geq n - k$), 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 $\ell \geq n - k$ 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.

$\square_{Theorem\ 1}$

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. $\square_{Theorem\ 2}$

3 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 $\mathcal{CARW}_{n,t}[1 \leq t \leq k < n-1, k\text{-IS}]$, for $x \geq \mathsf{max}(1, t+k-(n-2))$. The algorithm providing such an implementation is Algorithm 1, whose operation name is $\mathsf{propose}_x()$.

```
operation propose_x(v) is
(1)  view_i ← IS.write_snapshot_k(v);
(2)  VIEW[i] ← view_i;
(3)  wait(|{ j such that VIEW[j] ≠ ⊥}| = 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 x-SA in $\mathcal{CARW}_{n,t}[1 \leq t \leq k < n-1, k\text{-IS}]$ (code for p_i)

Theorem 3 *Let $x \geq \mathsf{max}(1, k+t-(n-2))$. 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

When it calls $\mathsf{propose}_x(v)$, 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 until it sees the views of at least $(n-t)$ 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 implements x-*SA* object in the system model $\mathcal{CARW}_{n,t}[1 \leq t \leq k < n-1, k$-*IS*$]$.

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 worst 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 obtain $V(1)$ 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 $V(2)$. 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 $x \geq 1$ (consensus) for $n-t > k$, and $x \geq k+t-(n-2)$ otherwise (i.e., when $n-t \geq k-x+2$), from which follows that $x \geq \mathsf{max}(1, k+t-(n-2))$, which completes the proof of the theorem. $\square_{Theorem\ 3}$

The next corollary is a re-statement of Theorem 3 for $x=1$.

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

4 Impossibility of t-Terminating k-Immediate Snapshot

We have shown in Section 3 that, with a k-IS object added to the model $\mathcal{CARW}_{n,t}[1 \leq t \leq k < n-1]$, it is possible to implement an x-SA object for $x \geq \mathsf{max}(1, k+t-(n-2))$. There are two cases.

- Case $k+t \leq n-1$.In this case, it means that consensus can be implemented, which is impossible as soon as there is at least one failure [20] (assumption of the model).
- Case $k+t > n-1$.In this case, it means that $k+t-(n-2)$-SA can be implemented. But, by the model assumption $k \leq n-2$, we have $t \geq k+t-(n-2)$. But it is known ([3, 17, 24]) that it is impossible to achieve $(k+t-(n-2))$-SA in $\mathcal{CARW}_{n,t}[1 \leq t \leq k < n-1]$.

So we have:

Theorem 4 *It is impossible to implement a* k-IS *object in the model* $\mathcal{CARW}_{n,t}[1 \leq t \leq k < n-1]$.

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 that consensus can be implemented from a k-IS object in $\mathcal{CARW}_{n,t}[1 \leq t < n/2, t \leq k \leq (n-1)-t, k\text{-IS}]$ (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 $\mathcal{CARW}_{n,t}[t \leq k \leq n-1, \mathbf{CONS}]$

Algorithm 2 describes a reduction of k-IS to consensus in $\mathcal{CARW}_{n,t}[0 < t \leq k \leq n-1, \text{CONS}]$. 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 $\mathcal{CARW}_{n,t}[t \leq n-1]$).
The behavior of a process p_i can be decomposed in three parts.

```
operation write_snapshot_k(v_i) is
(1)  REG[i] ← v_i;
(2)  wait (|j such that REG[j] ≠ ⊥}| ≥ n − k);
(3)  aux_i ← {⟨j, REG[j]⟩ such that REG[j] ≠ ⊥};
(4)  view_i ← CS.propose_1(aux_i);
(5)  if (⟨i, v_i⟩ ∈ view_i)
(6)     then return(view_i)
(7)     else aux_i ← IS.write_snapshot(v_i);
(8)          view_i ← view_i ∪ aux_i;
(9)          return(view_i)
(10) end if
end operation.
```

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

- When it invokes 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* $\mathcal{CARW}_{n,t}[0 < t \leq k \leq n-1, \text{CONS}]$.

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 the following three observations: (a) the fact that a process p_i assigns to $REG[i]$ the value it wants to deposit in the k-IS object, (b) the fact that this atomic variable is written at most once (line 1), and (c) the fact that the predicate $REG[j] \neq \perp$ 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 8).

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 $view_j$ returned by a process that terminates at line 9 includes $view$. It follows that $\forall\, p_i \in$ P6, $p_j \in$ P9, we have $view_i = view \subset view_j$.

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 consider the sets P6 and P9 defined above. There are four 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 belongs 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 the underlying object IS, we have (at line 7) $\langle i, -\rangle \in aux_j \Rightarrow aux_i \subseteq aux_j$ (and $\langle j, -\rangle \in aux_i \Rightarrow aux_j \subseteq aux_i$). Let $view$ be the set of pairs returned by the underlying 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.
- p_i belongs to P9, while p_j belongs to P6. By the agreement property of the underlying consensus object CS, it follows that $view_i$ and $view_j$ (line 4). Let V be the corresponding value of $view_i$ and $view_j$. As p_j belongs to P6, p_j outputs $view_j = V$. As by assumption $\langle i, v\rangle \in view_j$ i.e $\langle i, v\rangle \in V$, it follows that, when p_i executes line 5, we have $view_i = V$ at this point, and finds $\langle i, v\rangle \in view_i$. The predicate of line 5 is consequently satisfied and p_i belongs to P6. So this case is impossible.

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. $\square_{Theorem\ 5}$

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* $\mathcal{CARW}_{n,t}[0 < t < n/2, t \leq k \leq (n-1) - t]$.

6 When Consensus is Stronger than k-Immediate Snapshot

Section 3 investigated the power of k-IS to implement x-SA objects, namely x-SA can be implemented in $\mathcal{CARW}_{n,t}[1 \leq t \leq k < n-1, k\text{-IS}]$ where $x = \mathsf{max}(1, t + k - (n-2))$, see Theorem 3. As we have seen, considering the other direction, Section 5 has shown that k-IS can be implemented in $\mathcal{CARW}_{n,t}[1 \leq t \leq k < n-1, \text{CONS}]$ (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 $\mathcal{CARW}_{n,t}[n/2 \leq t \leq k < n-1]$.

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

Proof

The proof is by contradiction: we prove that if there were a t-terminating consensus algorithm $\mathcal{A}$ for n processes using k-IS objects in a system where $\lceil n/2 \rceil \leq t \leq k < n-1$, then we would have a read-write wait-free consensus algorithm for two processes (namely a 1-terminating consensus algorithm in $\mathcal{CARW}_{n,t}[n=2,t=1]$) what is impossible [13].

For this we define a simulation in which two (actual) processes Q_0 and Q_1 simulate algorithm $\mathcal{A}$ on $\{p_1, \cdots, p_n\}$ in such a way to achieve (wait-free) consensus between Q_0 and Q_1. The details of the simulation are given below, we only explain now only the main points.

In this simulation Q_0 and Q_1 simulate steps of disjoint sets of simulated processes Π_0 and Π_1. More precisely, Π_0 and Π_1 is a partition of $\{p_1, \cdots, p_n\}$ such that $|\Pi_0| \leq t$ and $|\Pi_1| \leq t$, and Q_0, simulates in a round-robin way steps of Π_0, while Q_1 simulates in a round-robin way steps of Π_1.

In this simulation, if Q_x for $x = 0$ or $x = 1$ is correct (i.e., does not crash), each simulated process in Π_x executes its sequence of operations (it is consequently correct in the simulated run). If Q_x crashes, its crash entails (in the simulated run) the crashes of all the processes in Π_x. Note that, as at most t simulated processes may crash in a simulated run, no process of Π_{1-x} crashes if all processes of Π_x crash. Hence the simulation has to work only in such failure environments for simulated processes. Finally, as Π_0 and Π_1 the sets of simulated processes are disjoint the simulation of the read and write operations are trivial, the core of the simulation lies in the simulation of the operation write_snapshot$_k()$ issued by a simulated process p_i.

Simulation of a k-IS object Algorithm 3 (a snapshot-based variant of [3]) implements an immediate snapshot object shared by n processes $\{p_1, .., p_n\}$. To this end, it uses an array $A[1..n]$ of snapshot objects, each containing up to n pairs (proc. id, value). (Algorithms building snapshot objects on top of read/write registers can found in the literature (e.,g., [1, 19].)

```
operation write_snapshot(i, v_i) is
(1)  r_i ← n + 1;
(2)  while true do
(3)      r_i ← r_i − 1;
(4)      A[r].update(i, v_i); set_i ← A[r].snapshot();
(5)      if (|set_i| = r_i) return(set_i) end if
(6)  end while.
```

Algorithm 3: Immediate snapshot object from snapshot objects (code of p_i)

Let us consider Algorithm 3, which implements an $(n-1)$-IS object on top of an array of snapshot objects $A[1..n]$. When a process p_i invokes the operation write_snapshot(i, v_i), it obtains a set of pairs set_i that satisfies the Immediate Snapshot properties, namely Termination, Self-inclusion, Validity and Containment. Each of the n processes asynchronously executes exactly x rounds where, from its local point of view, x is the exact number of processes that have deposited pairs in the snapshot object $A[x]$. The set of pairs returned by a process p_i is then the set set_i which contains the x corresponding pairs. If the simulator Q_0 simulates the previous code for the processes p_i, $1 \leq i \leq t$, and the simulator Q_1 does the same for the processes p_j, $t+1 \leq j \leq 2t$, they simulate an $(n-1)$-IS object.

But, while Algorithm 3 solves the case $k = n-1$, the aim is to address the general case $k < n-1$, where each set output by a k-IS object must contain at least $n-k$ pairs. Hence, the simulation consists in "extending" Algorithm 3 so that it works for $t \leq k \leq n-1$. To this end, the simulation of write_snapshot() described in Algorithm 3 is replaced by two operations which share the array $A[1..n]$. These operations denoted initsimul_k_is() and simul_k_is(), are described in Algorithm 4. The operation simul_k_is() executed by a simulator (lines 11-16) is the same as the operation write_snapshot() described in Algorithm 3. In order to simulate a k-IS object, some synchronization is needed, namely, before any simulator invokes simul_k_is(), at least one of the simulators has to invoke the operation

initsimul_k_is() with a set of pairs (j, v_j) of size $n-k$. More precisely, while the operation simul_k_is() allows a simulator Q_x to simulate the invocation of write_snapshot$_k$() by a process, the operation initsimul_k_is() allows it to simulate the invocation write_snapshot$_k$() not for only one, but for a set of $n-k$ processes. To this end, when it invokes simul_k_is(), the simulator Q_x first updates $A[n]$ for these $n-k$ simulated processes, and then takes a snapshot of $A[n]$ (line 5). Let set_x be the obtained set from this snapshot. This set contains at least $n-k$ pairs. So, if $|set_x| = n$, the simulator Q_x return set_x as output for each of the $n-k$ simulated processes and terminates the operation. If $|set_x| < n$, the simulator progresses to the next iteration step $r_x - 1$, and continues until $|set_x| = r_x$, which inevitably occurs.

```
operation initsimul_k_is(set) is      % set is a set of n − k pairs (i, v)
(1)  r_x ← n + 1;
(2)  while true do
(3)      r_x ← r_x − 1;
(4)      for each pair (i, v) ∈ set do A[r_x].update(i, v) end for;
(5)      set_x ← A[r_x].snapshot();
(6)      if (|set_x| = r_x) then
(7)          for each pair (i, v) ∈ set do out_x[i] ← set_x end for;
(8)          return(out_x)
(9)      end if
(10) end while.

operation simul_k_is(i, v_i) is
(11) r_x ← n + 1;
(12) while true do
(13)     r_x ← r_x − 1;
(14)     A[r_x].update(i, v_i); set_x ← A[r_x].snapshot();
(15)     if (|set_x| = r_x)then return(set_x) end if
(16) end while.
```

Algorithm 4: Operations used by Q_x ($x \in \{0,1\}$) to simulate a k-IS object shared by $n = 2t$ processes

We say that the operations initsimul_k_is() and simul_k_is() are *well-used* if:

- Q_0 invokes initsimul_k_is() at most once. The set parameter set of this invocation is a subset $\subseteq \{p_1, ...p_t\}$ of size $n-k$.
- For each simulated process $p_i \in \{p_1, ...p_t\}$, Q_0 invokes at most once simul_k_is() for p_i or initsimul_k_is(set) such that $p_i \in set$ and $set \subseteq \{p_1, ...p_t\}$ (but not both).
- The same is required for Q_1 where the set $\{p_1, ..., p_t\}$ is replaced by the set $\{p_{t+1}, ...p_{2t}\}$.

We trivially have the following lemma.

Lemma 1 *Considering an execution in which one simulator Q_0 or Q_1 may crash, if Q_0 and Q_1 well-use* initsimul_k_is() *and* simul_k_is()*, then their corresponding outputs satisfy the properties of immediate snapshot, namely : Termination, Self-inclusion, Validity and Containment.*

Lemma 2 *Considering an execution in which one simulator Q_0 or Q_1 may crash, if Q_1 and Q_0 well-use* initsimul_k_is() *and* simul_k_is()*, the invocation of* initsimul_k_is(set) *by Q_0 (resp. Q_1) returns an array out_0 (resp. out_1) such that for each process $p_\ell \in set$, $out[\ell]$ contains at least $n-k$ pairs.*

Proof If the simulator Q_0 invokes initsimul_k_is(set), when it executes the loop for some value of r, it sequentially executes $|set| = n-k$ updates on the array $A[r]$ (one for each simulated processes of set, (lines 4) and then takes a snapshot (line 5). This snapshot contains at least $|set| = n-k$ elements. Then either Q_0 proceeds to the next round or writes in out_0 a set of at least $(n-k)$ pairs. The proof of termination is the same as the one for an immediate snapshot object (see [3, 23]). $\square_{Lemma\ 2}$

Lemma 3 *Considering a 1-terminating execution in which one simulator Q_0 or Q_1 may crash, if (1) both Q_1 and Q_0 well-use the operations* initsimul_k_is() *and* simul_k_is()*, and (2) the first invocation of either Q_0 or Q_1 (may be both) is* initsimul_k_is()*, and (3) if the first invocation of Q_x ($x \in \{0,1\}$) is not* initsimul_k_is()*, its first invocation of* simul_k_is() *occurs after the first invocation of* initsimul_k_is() *by Q_{1-x}, the returned values (either by* initsimul_k_is() *or by* simul_k_is()*) satisfy the properties defining a k-*IS *object.*

Proof By Lemma 1, the returned values satisfy the immediate snapshot properties. By Lemma 2, the returned values of initsimul_k_is() contains at least $(n-k)$ values. Let us now consider an invocation of simul_k_is() issued say by Q_x. By assumption, this invocation is always preceded by an invocation of initsimul_k_is() either by Q_x or by Q_{1-x}. After this first invocation of initsimul_k_is(), one of the snapshot object $A[r]$ contains r pairs, where r is such that $n \geq r \geq n-k$ and no snapshot object $A[r']$, $r' > r$, will contain less than r pairs. So, when Q_x invokes simul_k_is(), it obtains a (set) snapshot value at the latest for $r_x = n-k+1$, the size of which is equal to r_x. $\square_{Lemma\ 3}$

Let P_0 and P_1 be a partition of $\{p_1, \cdots, p_n\}$:
$|P_0| = |P_1| = t$, $\{p_1, \cdots, p_n\} = P_0 \cup P_1$, and $P_0 \cap P_1 = \emptyset$.

Additional shared object:
$REG[0..1]$: array of values in $\{\texttt{done}, \texttt{notdone}\}$, initialized to notdone.
Local variables: $prop_x$ initialized to $\emptyset$.

(1) **for all** $p_j \in P_x$: initialize the input value v_j of p_j to the input value of Q_x;
(2) **repeat forever**
(3) **for each** $p_i \in P_x$ in a round robin way **do**
(4) **if** next operation of p_i is write_snapshot$_k(v)$
(5) **then if** $(REG[x] = \texttt{done} \ \vee \ REG[1-x] = \texttt{done})$ **then**
(6) **then** $result \leftarrow$ simul_k_is(i, v_i)
simulation of the operation write_snapshot$_k(v)$ issued by p_i
with $result$ has returned value
(7) **else** $prop_x \leftarrow prop_x \cup \{(i, v_i)\}$;
(8) **if** $|prop_x| = n-k$ **then**
(9) $out_x \leftarrow$ initsimul_k_is$(prop_x)$; $REG[x] \leftarrow$ done;
(10) **for each pair** $(j, w) \in prop_x$ **do**
(11) $result \leftarrow out_x[j]$
simulation of the operation write_snapshot$_k(v)$ issued by p_i
with $result$ has returned value
(12) **end for**;
(13) **end if**
(14) **end if**
(15) **else** simulate the next operation of p_i;
(16) **if** the processes p_1, ..., p_n are engaged in a consensus algorithm
and p_i decides d in this step **then** Q_x decides d **end if**
(17) **end if**
(18) **end repeat**.

Algorithm 5: Simulation of $\mathcal{A}$ by Q_x ($x \in \{0,1\}$) for $n = 2t$

Simulation of $\mathcal{A}$ by Q_x ($x \in \{0,1\}$) Algorithm 5 presented below allows the pair of simulators Q_0 and Q_1 to simulate a k-IS object shared by the simulated processes p_1, ..., p_n.[6] The simulator processes Q_0 and Q_1 manage the following variables.

[6]The case of several k-IS objects for various values of k ($t \leq k < n-1$) can be easily addressed by multiplexing the simulation on these objects. To this end the shared variables $REG[0]$, $REG[1]$, and the local variables $prop_x$ must be replaced by arrays, with one entry for each simulated k-IS object.

- $REG[0,1]$ is an array made up of two atomic read/write registers associated with the simulated k-IS object. $REG[x]$ is written by Q_x and read by both the simulators Q_x and Q_{1-x}. $REG[x] =$ done indicates that Q_x has invoked initsimul_k_is().
- $prop_x$ is a local variable of Q_x containing the values not yet written to the k-IS object by the simulated process $p_i \in P_x$ (line 7).

A simple observation of Algorithm 5 shows that the assumption stated in Lemma 3 are satisfied, from which we have the following lemma.

Lemma 4 *Considering a 1-terminating execution of Q_0 and Q_1 of the simulation Algorithm 5, the values returned (by* simul_k_is() *or* initsimul_k_is()*) satisfy the properties defining a k-IS object.*

Concerning the liveness of the simulation, it remains to show that if the simulator Q_x is correct it executes an infinite number of steps of its associated simulated processes.

Let us remark first that, in an execution of the processes p_1, ..., p_n, as a k-IS object KIS contains pairs from at least $(n-k)$ processes. Hence, for a process –that invokes KIS.write_snapshot$_k()$– to be able to terminate, $(n-k)$ processes must have invoked the operation KIS.write_snapshot$_k()$. Let us say that a simulated process is "*in front of KIS*" if its next step is KIS.write_snapshot$_k()$. If a process is in front of KIS, eventually at least $(n-k)$ simulated processes are in front of KIS. But, as we consider that at most t simulated processes may crash, we have the following stronger property.

Property 1 *Let us consider a t-terminating execution of $\mathcal{A}$ (simulated by Q_0 and Q_1) in which a simulated process halts just before it invokes the operation KIS.write_snapshot$_k()$. If a simulated process of P_x is in front of KIS then there is a time after which at least $n-k$ simulated processes of P_x are in front of KIS.*

Proof Considering the simulator Q_x, as the execution is t-resilient, let us assume that the processes in P_{1-x} in front of KIS crashed. So, all the simulated processes in P_x are correct and have to get an answer of the KIS. Hence, there is at least $(n-k)$ processes of P_x that are in front of KIS. $\square_{Property\ 1}$

Lemma 5 *If the simulator Q_x does not crash it simulates an infinite number of steps of the processes in P_x.*

Proof From Algorithm 5, Q_x forever tries to simulate a step of processes in P_x. If the next operation of the (simulated) process p_i is not a write_snapshot$_k()$ on the KIS object (line 15), then Q_x succeeds in simulating the step of the simulated process p_i.

If the next operation of the (simulated) process p_i is a write_snapshot$_k(v)$ on the KIS object (line 4) then either (1) $REG[x]$ or $REG[1-x]$ (line 5) is equal to done, or (2) $\langle i, v\rangle$ is added to $prop$ (line 7).

- In the first case, Q_0 or Q_1 has already simulated (by executing initsimul_k_is(), line 9) the operation write_snapshot$_k()$ on the KIS object for $n-k$ (simulated processes) in P_0 or in P_1. Q_x is able to simulate the step write_snapshot$_k(v)$ of p_i by executing simul_k_is() (line 9). Hence, Q_x succeeds in simulating the step of the simulated process p_i.
- In the second case, the pair $\langle i, v\rangle$ is added to $prop$ (line 7). There are two sub-cases.
  - If Q_x gets $n-k$ processes in P_x (line 8) in front of the k-IS object KIS, it invokes initsimul_k_is() and is able to simulate the step of this process p_i and of the other processes in P_x in front of the KIS object. Hence, Q_x succeeds to simulate the step of the simulated process p_i(line 11).

– If less than $n-k$ processes of P_x are in front of KIS, no step of p_i is simulated. Remark that in the next loop iteration (line 2), Q_x tries again to simulate the step of p_i.

By Property 1, there is a time after which $n-k$ processes of P_x are in front of the KIS object or Q_{1-x} has executed initsimul_k_is() and has written done in $REG[1-x]$.

After this occurred, in the next loop (line 2), when Q_x tries again to simulate the step of one of the processes in front of the KIS object we have either either (1) $REG[1-x]$ (line 5) is equal to done, or (2) $prop$ (line 7) contains $n-k$ processes. As we have already seen, in this case Q_x is able to simulate the step of p_i. $\square_{Lemma\ 5}$

So as soon as Q_x is correct, it simulates an infinite number of steps of processes in P_x. It is trivial to show that a *terminating* execution of Q_0 and Q_1 of Algorithm 5 simulates a t-resilient execution of the consensus algorithm $\mathcal{A}$ for a set of n processes $\{p_1, ..., , p_n\}$. As the input value of the processes in P_x are the input values of Q_x, then the validity, agreement and termination of $\mathcal{A}$ provide us with the validity, agreement and termination of the executions of Q_0 and Q_1, and consequently Algorithm 5 is a 1-terminating consensus algorithm for two processes, which is know to be impossible [20].

To extend the result to $2t > n$, we partition $\{p_1, \cdots, p_n\}$ in 3 sets P_0, P_1, D such that $|P_0| = n-t$, $|P_1| = n-t$, $|D| = 2t-n$. Then, we run the previous simulation Algorithm 5 where all the processes in D are initially crashed, Q_0 simulates the set of processes in P_0, and Q_1 simulates the processes in P_1. Algorithm 5 would be a 1-terminating consensus algorithm for two processes which is impossible. $\square_{Theorem\ 7}$

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 Figure 6.

- 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 $x = \max(1, t+k-(n-2))$, 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 $\lceil n/2 \rceil \leq t \leq k < n-1$ (Zone C).

Stated in a more operational way, these results exhibit the price of the synchronization hidden in a k-IS object, which requires that the view returned to a process contains at least $(n-k)$ pairs. Recall that 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.

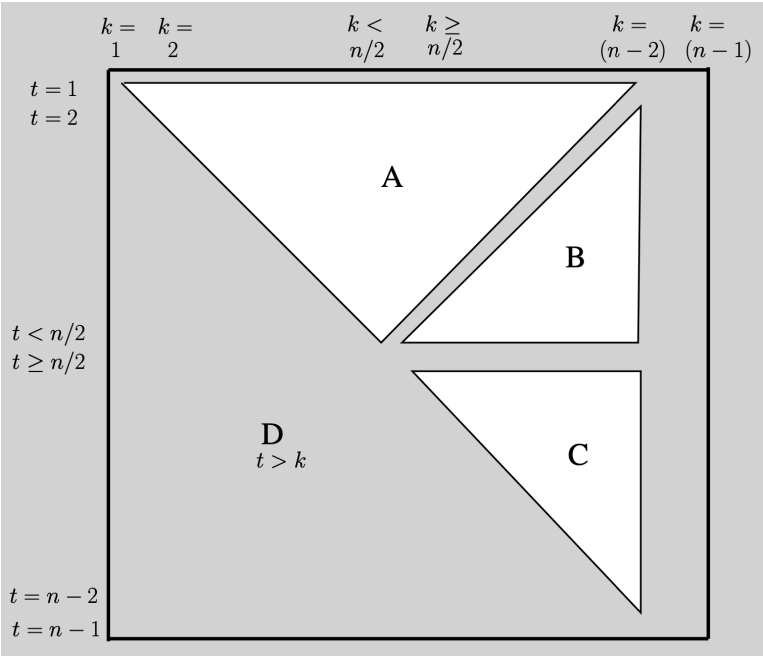

Figure 6: Summarizing the results.

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)?

Discussion. The liveness of algorithms described in the article explicitly depends on the interplay between the failure pattern (captured as the maximal number t of processes that may crash) and the fact that the set of pairs returned from a k-IS object to a process must contain at least $(n-k)$ pairs from different processes. Said in another way, differently from x-set agreement and immediate snapshot, k-immediate snapshot is both an agreement and a synchronization object.

This raises the following question: Is there another meaningful generalization of immediate snapshot (i.e., not suffering from the synchronization constraint) or is synchronization-free immediate snapshot restricted to the case $t = n - 1$?

Acknowledgments

The authors want to thank the referees for their constructive comments that helped improve both the presentation and the content of the article, and motivated the discussion at the end of the conclusion.

This work has been partially supported by the French ANR project DESCARTES (16-CE40-0023-03) devoted to modular structures in distributed computing, the French ANR project ByBLoS (ANR-20-CE25-0002-01) devoted to the modular design of building blocks for large-scale trustless multi-users applications, and the UNAM-PAPIIT projects IN107714 and IN106520.

References

[1] Afek Y., Attiya H., Dolev D., Gafni E., Merritt M. and Shavit N., Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873-890 (1993)

[2] Attiya H. and Ellen F., *Impossibility results for distributed computing*. Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool, 162 pages (2014)

[3] Borowsky E. and Gafni E., Immediate atomic snapshots and fast renaming. *Proc. 12th ACM Symp. on Principles of Distributed Computing (PODC'93)*,ACM Press, pp. 41-50 (1993)

[4] Borowsky E. and Gafni E., Generalized FLP impossibility results for t-resilient asynchronous computations. *25th ACM Symp. Theory of Comp.*, ACM Press, pp. 91-100 (1993)

[5] Borowsky E. and Gafni E., A simple algorithmically reasoned characterization of wait-free computations. *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC'97)*, ACM Press, pp. 189-198 (1997)

[6] Borowsky E., Gafni E., Lynch N. and Rajsbaum S., The BG distributed simulation algorithm. *Distributed Computing*, 14:127-146 (2001)

[7] Castañeda A., Rajsbaum S., and Raynal M., Unifying concurrent objects and distributed tasks: interval-linearizability. *Journal of the ACM*, 65(6), Article 45, 42 pages (2018)

[8] Chaudhuri S., More choices allow more faults: set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132-158 (1993)

[9] Delporte C., Fauconnier H., Rajsbaum S., and Raynal M., t-Resilient immediate snapshot is impossible. *Proc. 23nd Int'l Colloquium on Structural Information and Communication Complexity (SIROCCO'16)*, Springer LNCS 9988, pp. 177-191 (2016)

[10] Delporte C., Fauconnier H., Rajsbaum S., and Raynal M., k-Immediate snapshot and x-set agreement: how are they related? *Proc. 22nd International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'20)*, Springer LNCS 12514, pp. 97-112 (2020)

[11] Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382 (1985)

[12] Gafni E. and Rajsbaum S., Distributed programming with tasks. *Proc. 14th International Conference Principles of Distributed Systems (OPODIS'10)*, Springer LNCS 6490, pp. 205-218 (2010)

[13] Herlihy M. P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149 (1991)

[14] Herlihy M.P., Kozlov D., and Rajsbaum S., *Distributed computing through combinatorial topology*, Morgan Kaufmann/Elsevier, 336 pages, ISBN 9780124045781 (2014)

[15] Herlihy M., Rajsbaum S., and Raynal M., Power and limits of distributed computing shared memory models. *Theoretical Computer Science*, 509:3-24 (2013)

[16] Herlihy M. P. and Shavit, N., A simple constructive computability theorem for wait-free computation. *26th ACM Symposium on Theory of Computing*, ACM Press, pp. 243-252 (1994)

[17] Herlihy M. P. and Shavit, N., The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858-923 (1999)

[18] Herlihy M. P. and Wing J. M., Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463-492 (1990)

[19] Imbs D. and Raynal M., Help when needed, but no more: efficient read/write partial snapshot. *Journal of Parallel and Distributed Computing*, 72(1):1-13 (2012)

[20] Loui M. and Abu-Amara H., Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163-183, JAI Press (1987)

[21] Neiger G., Set-linearizability. Brief announcement in *Proc. 13th ACM Symposium on Principles of Distributed Computing (PODC'94)*, ACM Press, page 396 (1994)

[22] Rajsbaum S., Iterated shared memory models. *Proc. 9th Latin American Symposium Theoretical Informatics (LATIN'10)*, Springer LNCS 6034, pp. 407-416 (2010)

[23] Raynal M., *Concurrent programming: algorithms, principles and foundations*. Springer, 515 pages, ISBN 978-3-642-32026-2 (2013)

[24] Saks M. and Zaharoglou F., Wait-free k-set agreement is impossible: the topology of public knowledge. *SIAM Journal on Computing*, 29(5):1449-1483 (2000)

[25] Saraph V., Herlihy M. and Gafni E., Asynchronous computability theorems for t-resilient systems. *Proc. 30th International Symposium on Distributed Computing (DISC'16)*, Springer LNCS 9888, pp. 428-441 (2016)

[26] Taubenfeld G., Coordination without prior agreement. *Proc. 36th ACM Symposium on Principles of Distributed Computing (PODC'17)*, ACM Press, pp. 325-334 (2017)