Theoretical Computer Science 410 (2009) 1328–1335

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

From adaptive renaming to set agreement

Eli Gafni [a], Achour Mostéfaoui [b], Michel Raynal [b,*], Corentin Travers [b]

[a] Department of Computer Science, UCLA, Los Angeles, CA 90095, USA

[b] IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

ARTICLE INFO

Keywords:
Adaptive renaming
Asynchronous algorithm
Atomic register
Atomic snapshot
Participating process
Reduction
Set agreement
Shared object
t-resilient algorithm
Wait-free algorithm

ABSTRACT

The adaptive M-renaming problem consists of providing processes with a new name taken from a name space whose size M depends only on the number p of processes that participate in the renaming (and not on the total number n of processes that could ask for a new name). The k-set agreement problem allows each process that proposes a value to decide a proposed value in such a way that at most k different values are decided. In an asynchronous system prone to up to t process crash failures, and where processes can cooperate by accessing atomic read/write registers only, the best that can be done is a renaming space of size $M = p+t$. In the same setting, the k-set agreement problem cannot be solved when $t \geq k$.

This paper focuses on the way a solution to the adaptive renaming problem can help in solving the k-set agreement problem when $t \geq k$. It has two contributions. Considering the case $k = t$ ($1 \leq t < n$), the first contribution is a t-resilient algorithm that solves the k-set agreement problem from any adaptive $(p + k - 1)$-renaming algorithm. The second contribution considers the case $k < t$. It shows that there is no such wait-free algorithm when $k < n/2$ (wait-free means $t = n-1$). So, while a solution to the adaptive $(p+k-1)$-renaming problem allows t-resiliently solving the k-set agreement problem despite $t = k$ failures, when $k < t$ such an additional power becomes useless for the values of $n > 2k$ (i.e. adaptive $(p + k - 1)$-renaming allows progressing from $k > t$ to $k = t$, but does not allow bypassing the "$k = t$" frontier when $n > 2k$).

1. Introduction

Renaming and set agreement. Renaming and set agreement are among the basic problems that lie at the core of computability in asynchronous systems prone to process crashes. The *renaming* problem (introduced in [3]) consists of designing an algorithm that allows processes (that do not crash) to obtain new names from a new name space that is as small as possible, and such that no two processes acquire the same new name. In the following M denotes the size of the new name space, and a corresponding algorithm is called an M-renaming algorithm.

The renaming problem has initially been introduced from a theoretical point of view [3]. The aim was to state a non-trivial coordination problem that, unlike the consensus problem, can be solved despite process crashes. It appeared then that the renaming problem is an instance of a more general resource allocation problem (a new name is a "slot/token/etc." that can be attributed to a single process) [7].

A *wait-free* algorithm is an algorithm that allows each process that does not crash to terminate in a finite number of computation steps, whatever the behavior of the other processes (i.e., despite the fact that all the other processes are extremely slow, or even have crashed) [17]. It has been shown that, in a system of n processes that can communicate through

* Corresponding author. Tel.: +33 299 84 71 88.
E-mail addresses: eli@cs.ucla.edu (E. Gafni), achour@irisa.fr (A. Mostéfaoui), raynal@irisa.fr (M. Raynal), ctravers@irisa.fr (C. Travers).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.05.016

atomic read/write registers only, the smallest new name space that a wait-free renaming algorithm can produce is bounded from below by $M = 2n-1$ [20]. More generally, in an asynchronous system where up to t processes may crash, the smallest value of M is $n + t$ (the wait-free case corresponds to $t = n - 1$).

The k-set agreement problem has been introduced in [11]. It is a paradigm of coordination problems encountered in distributed computing and is defined as follows. Each process is assumed to propose a value. The problem consists in designing an algorithm such that (1) each process that does not crash decides a value (termination), (2) a decided value is a proposed value (validity), and (3) no more than k different values are decided (agreement). (The well-known consensus problem is the same as the 1-set agreement problem.) The parameter k can be seen as the coordination degree (or the difficulty) associated with the corresponding instance of the problem. The smaller k, the more coordination among the processes: $k = 1$ means the strongest possible coordination, while $k = n$ means no coordination.

It has been shown in [9,20,29] that, in an asynchronous system made up of processes that communicate through atomic registers only, and where up to t processes may crash, there is no wait-free k-set agreement algorithm for $k \leq t$. However, when $k > t$ the problem can be trivially solved (a predefined set of k processes write their proposal, and a process decides the first proposal it reads).

Randomized or failure detector-based algorithms have been proposed to circumvent the previous impossibility result [18, 23,24]. An algorithm that wait-free solves the $(n - 1)$-set agreement problem in a system of n crash-prone asynchronous processes from $(2n - 2)$-renaming objects is described in [13].

Asynchronous computability. An important issue in fault-tolerant asynchronous computing is the determination of the respective power of an object type with respect to another object type (an object type corresponds to a problem). This question has received a lot of attention, mainly in the context of the consensus problem where a major advance has been the introduction of the *consensus number* notion[1] that allows ranking the synchronization power of base object types (atomic registers, queues, test&set objects, compare&swap objects, etc.) with respect to the consensus problem [17]. This has given rise to the well-known Herlihy's hierarchy [17].

Due to its very definition, the consensus number notion is irrelevant for studying the respective power of object types (problems) that are too weak to solve the consensus problem. These problems are usually called *subconsensus* types/problems. Renaming and k-set agreement (for $k \neq 1$) are subconsensus problems. So, an important issue of asynchronous computability consists in establishing connections (or absence of connection) between subconsensus problems. Several results in that direction have recently been established (e.g., [2,14–16,25,27]). In a very interesting way, it is shown in [15] (using arguments from combinatorial topology) that the renaming problem is strictly less powerful than the set agreement problem in the round-by-round model of asynchronous computation (as defined in [12]).

Adaptive renaming. A renaming algorithm is *adaptive* if the size of the new name space depends only on the number p of processes that ask for a new name (and not on the total number n of processes). Several adaptive algorithms have been designed such that the size of the new name space is $M = 2p - 1$ (e.g., [4,5,8]) (these adaptive algorithms are consequently optimal with respect to the size of the new name space [20]). This means that if "today" p' processes acquire new names, their new names belong to the interval $[1..2p' - 1]$. If "tomorrow" p'' additional processes acquire new names, these processes will have their new names in the interval $[1..2p - 1]$ where $p = p' + p''$.

Recently, with the aim of circumventing the $M = 2p - 1$ lower bound, researchers have investigated the use of base objects stronger than atomic registers in order to solve the renaming problem. Following this line of research, it has been shown in [25] that, as soon as k-test&set objects can be used, the adaptive renaming problem can be wait-free solved with a new name space the size of which is size $M = 2p - \lceil \frac{p}{k} \rceil$.[2] Among the processes that access it, a k-test&set object ensures that at least one and at most k processes obtain the value 1 (they win), while all the other processes obtain the value 0 (they lose) (the usual test&set object is a 1-test&set object). It has also been shown in [14] that the adaptive renaming problem can be wait-free solved with a new name space of size $M = p + k - 1$ as soon as k-set agreement objects can be used. According to the base objects they use, respectively, both algorithms are optimal [16] with respect to the size of their new name space.

Content of the paper. When we consider asynchronous systems where the processes communicate through atomic registers only, the k-set agreement problem can be (easily) solved when $k > t$, and is impossible to solve when $k \leq t$. Moreover, as shown in [15] (and indicated above) the non-adaptive version of the renaming problem is strictly less powerful than the k-set agreement problem, and is consequently useless to solve that problem. So, an important question that remains to be answered is the following one: "Can solutions to the adaptive renaming problem help solving the k-set agreement problem?" To answer that question, the paper considers two cases: $k = t$, and $k < t$. (Let us recall that an algorithm is t-*resilient* if it always preserves its safety and liveness properties when no more than t processes commit failures. The notion of t-resilience boils down to the wait-free notion when $t = n - 1$.)

- Considering first the case $k = t$, the paper presents a t-resilient algorithm that solves the k-set agreement problem from atomic registers and an adaptive $(p + k - 1)$-renaming object. This has two consequences:

[1] The consensus number of an object type (defined by a sequential specification) is the maximum number of processes for which objects of that type plus atomic registers can wait-free solve the consensus problem.

[2] The adaptive renaming algorithm presented in [25] is actually based on k-set agreement objects. But it can easily be observed that these objects can be replaced by k-test&set objects without affecting the behavior of the renaming algorithm.

  - Enriching an asynchronous read/write system with an adaptive $(p + k - 1)$-renaming algorithm allows progressing from $k > t$ to $k = t$.
  - Differently from the non-adaptive renaming problem, the adaptive $(p + k - 1)$-renaming problem and the k-set agreement problem are wait-free equivalent (a wait-free algorithm going from the k-set agreement problem to the adaptive $(p + k - 1)$-renaming is presented in [14]).
- The paper then considers the case $k < t$. It establishes a lower bound for that case, namely, there is no algorithm (based on atomic registers and adaptive $(p+k-1)$-renaming objects) that can wait-free solve the k-set agreement object when $k < n/2$.

So, while a solution to the adaptive $(p + k - 1)$-renaming problem allows solving the k-set agreement problem despite $t = k$ failures, if $k < t$ such an additional power is useless when $k < n/2$: an adaptive $(p + k - 1)$-renaming object allows progressing from $k > t$ to $k = t$, but does not allow bypassing the "$k = t$" frontier when $n > 2k$. (Proving – or disproving – that, when $k < t$, the result still holds when $n/2 \leq k < n - 1$ remains an open problem. We conjecture the impossibility result is still true.)

After presenting the computation model (Section 2), the paper is composed of two sections. Section 3 considers the case $k = t$ and presents the t-resilient algorithm. Section 4 considers the case $k < t$ and presents the lower bound result. Finally, Section 5 concludes the paper.

2. Basic computation model

Process model. We consider systems made up of n asynchronous processes $p_1, \ldots, p_n$, where the integer i is the index of p_i. Π denotes the set of indexes, i.e. $\Pi = \{1, \ldots, n\}$. *Asynchronous* means that there is no bound on the time it takes for a process to execute a computation step. A process may crash (halt prematurely). After it has crashed a process executes no step. A process executes correctly its algorithm until it possibly crashes. The integer t, $0 \leq t < n$, denotes an upper bound on the number of processes that may crash; t is known by the processes. A process that does not crash in a run is *correct* in that run; otherwise, it is *faulty* in that run.

Communication model. The processes cooperate by accessing atomic read/write registers. *Atomic* means that each read or write operation appears as if it has been executed instantaneously at some time between its begin and end events [21,22]. Each atomic register is a one-writer/multi-readers (1WnR) register. This means that a single process (statically determined) can write it. Atomic registers are denoted with uppercase letters. The atomic registers are structured into arrays. $X[1..n]$ being such an array, $X[i]$ denotes the register of that array that p_i only is allowed to write. A process can have local registers. Such registers are denoted with lowercase letters with the process index appearing as a subscript (e.g., $winner_i$ is a local register of p_i).

The processes are provided with an atomic snapshot operation [1] denoted snapshot(X), where $X[1..n]$ is an array of atomic registers. It allows a process p_i to atomically read the whole array. This means that the execution of a snapshot() operation appears as if it has been executed instantaneously at some point in time between its begin and end events. Such an operation can be built from 1WnR atomic registers [1]. To our knowledge the best snapshot() implementation requires $O(n \log(n))$ read/write operations on base atomic registers [6].

Notions of t-resilience and wait-freedom. As indicated in the introduction, an algorithm is t-resilient if it copes with up to t process failures. In our context, this means that it satisfies its safety and liveness (termination) properties despite up to t process crashes. A wait-free algorithm is an $(n - 1)$-resilient algorithm.

Adaptive renaming. In the renaming problem, each process p_i has an initial name denoted id_i. Differently from the process indexes, id_i is initially known only by p_i. These names are from a very large name space, i.e., $\max(id_1, \ldots, id_n) >> n$. A renaming algorithm is *adaptive* with respect to the size of its new name space, if that size depends on the number of processes that actually participate in the renaming algorithm. A process participates in an algorithm as soon as it has written an atomic register used by that algorithm. Let us remark that an adaptive renaming algorithm cannot systematically assign the new name i to p_i. This is because, if only p_n wants to acquire a new name, the new name space is $[1..n]$, which depends on the number of processes instead of depending on the number of participating processes (here a single process). More generally, the following symmetry requirement is usually considered for the adaptive renaming problem [7]: the code executed by p_i with name id is the same as the code executed by process p_j with name id. This means that the process indexes can be used only for addressing purposes.

As indicated in the introduction, if p processes participate in an adaptive renaming algorithm based on atomic registers only, the best that can be done is a name space of size $M = 2p - 1$. More generally, let rename(id_i) be the operation issued by a process p_i to acquire a new name. The adaptive renaming problem is defined by the following properties:

- Termination. If a correct process invokes rename(), it obtains a new name.
- Uniqueness. No two processes obtain the same new name.
- Validity. A new name belongs to $[0..f(p)]$, where f is an increasing integer function and p $(1 \leq p \leq n)$ is the number of processes that have invoked rename().

This means that if, at time τ', p' processes have invoked rename() and obtained new names, these names belong to the set $[0..f(p')]$. Moreover (as noted in the introduction), if at a later time $\tau'' > \tau'$ (when each of the p' previous processes has

```
operation kset_propose(v_i):
(01)  PROP[i] ← v_i;
(02)  new_name_i ← rename(id_i);
(03)  RENAMED[i] ← 1 if new_name_i ≤ t, 0 otherwise;
(04)  repeat renamed_i ← snapshot(RENAMED) until |{j : renamed_i[j] ≠ ⊥}| ≥ (n − t);
(05)  let winners_i = {j : renamed_i[j] = 1};
(06)  if winners_i ≠ ∅ then ℓ_i ← any value ∈ winners_i
(07)                   else let set_i = {j : PROP[j] ≠ ⊥ ∧ renamed_i[j] = ⊥};
(08)                        ℓ_i ← any value ∈ set_i
(09)  end if;
(10)  return(PROP[ℓ_i])
```

Fig. 1. From $(p + k - 1)$-renaming to k-set agreement, for $k = t$, $\forall t$ (code for p_i).

crashed or obtained a new name), p'' additional processes invoke rename(), their new names belong to the set $[0..f(p)]$ where $p = p' + p''$. More generally, adaptive M-renaming is when $M = f(p)$.

3. From an adaptive $(p + k - 1)$-renaming to k-set agreement

Considering an asynchronous system made up of n processes, where up to t $(1 \leq t < n)$ of them may crash, cooperating through 1WnR one-write atomic registers, plus an adaptive $(p + k - 1)$-renaming object, this section presents and proves correct an algorithm that builds a k-set agreement object, when $k = t$ and at least $n - t$ correct processes participate in the k-set agreement algorithm.

Default value. The value $\perp$ denotes a default value that can appear only in the algorithms described in the paper.

3.1. Principles and description of the t-resilient algorithm

The principle of the transformation algorithm rests on two simple ideas.

1. First, use the underlying adaptive renaming object to partition the participating processes into two groups: the processes the name of which is smaller or equal to t (the winners); and the processes the name of which is greater than t (the losers). So, there are at most t winners.
2. Then, direct a process p_i to decide a value proposed by a winner. If p_i does not see winner processes, direct it to decide the value proposed by a process that has proposed a value but not yet obtained a new name.

To make these ideas operational, the shared memory is composed of two arrays of 1WnR one-write atomic registers.

- The array $PROP[1..n]$, initialized to $[\perp, \ldots, \perp]$, is such that $PROP[i]$ will contain the value (denoted v_i) proposed by p_i to the set agreement problem. A process p_i becomes *participating* as soon as $PROP[i] \neq \perp$.
- The aim of the array $RENAMED[1..n]$, also initialized to $[\perp, \ldots, \perp]$, is to allow the processes to benefit from the renaming object. When a process p_i has obtained a new name, $RENAMED[i]$ is set to 1 if its new name is smaller or equal to t (p_i is then a winner), while $RENAMED[i]$ is set to 0 if p_i is a loser. It trivially follows that $RENAMED[i] \neq \perp$ means that p_i has acquired a new name.

The behavior of a process p_i is described in Fig. 1. A process p_i invokes $\text{kset\_propose}_t(v_i)$ where v_i is the value it proposes to the k-set agreement problem. It decides a value when it executes the return(v) statement (line 10) where v is the value it decides. The way it implements the previous design ideas can be decomposed in two stages.

1. The first stage is composed of the lines 01–04. After it has deposited its proposal (line 01), obtained a new name (line 02), and updated $RENAMED[i]$ accordingly (line 03), a process p_i atomically reads the whole array *RENAMED* (using the snapshot() operation) until it sees that at least $n - t$ processes have acquired new names (line 04).
2. The second stage, composed of the lines 05–10, is the decision stage. If p_i sees a winner, it decides the value proposed by that winner process (lines 05, 06 and 10). If p_i sees no winner, it decides the value proposed by a process that (from its point of view) has not yet obtained a new name.

3.2. Proof of the algorithm

The proof considers that (1) $k = t$, i.e. the size of the new name space of the underlying adaptive renaming is $M = p+t-1$ when p processes participate, and (2) at least $(n - t)$ correct processes participate in the k-set agreement problem.

Notation and observation. Let RENAMED_i be the last value of $renamed_i$ when p_i exits the **repeat** loop at line 04. As a process p_x writes $RENAMED[x]$ at most once, we have $\text{RENAMED}_i[x] \neq \perp \wedge \text{RENAMED}_j[x] \neq \perp \Rightarrow \text{RENAMED}_i[x] = \text{RENAMED}_j[x]$. Let us define $\text{RENAMED}_i \leq \text{RENAMED}_j$ as $\forall x : \text{RENAMED}_i[x] \neq \perp \Rightarrow \text{RENAMED}_i[x] = \text{RENAMED}_j[x]$. Due to the atomicity property of

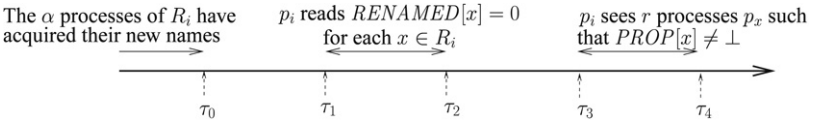

Fig. 2. Timing scenario.

the snapshot() operation (line 04), we have $\forall i, j$: $\text{RENAMED}_i \leq \text{RENAMED}_j \vee \text{RENAMED}_j \leq \text{RENAMED}_i$ (this is sometimes called the *containment* property provided by the snapshot() operation).

Lemma 1. *Let p_i be a process that decides. If $winners_i = \emptyset$ at line* 06*, then $set_i \neq \emptyset$ (when p_i executes line* 08*).*

Proof. Let p_i be a process that decides and has previously executed line 07. That process is such that $\forall x \in \Pi$: $\text{RENAMED}_i[x] = \perp$ or 0. Let $R_i = \{x : \text{RENAMED}_i[x] = 0\}$, and $\alpha = |R_i|$. Moreover, let $r = |\{x : PROP[x] \neq \perp\}|$ where the value of $PROP[x]$ is the value read by p_i at line 07. (See Fig. 2, where the time instants are such that $\tau_0 < \tau_3 < \tau_4$.) We show that $\alpha < r$, from which the claim follows (namely, there is a process p_y such that $PROP[y] \neq \perp \wedge \text{RENAMED}_i[y] = \perp$ when p_i executes line 07).

1. Let us first consider the processes p_x of the set R_i, i.e., such that $\text{RENAMED}_i[x] = 0$. These processes have obtained new names in a name space $[1..M]$ before time τ_0. We can conclude from the text of the algorithm that the new name obtained by each of these processes p_x (a loser) is such that $new\_name_x > t$ (lines 02 and 03). As there are α such processes we have $t + \alpha \leq M$.
2. Let ρ be the number of processes that started participating in the renaming before τ_0. We have seen (item 1) that M is the greatest name obtained by a process of R_i and that name has been obtained before τ_0. As the renaming algorithm is adaptive, we have $M \leq \rho + t - 1$.
3. As the ρ processes started participating in the renaming before before τ_0, they updated their entry in $PROP$ to a non-$\perp$ value before τ_0, and consequently we have $\rho \leq r$.
4. It follows from the previous items that $t + \alpha \leq M \leq \rho + t - 1 \leq r + t - 1$, from which we conclude $\alpha < r$, that terminates the proof of the lemma. □Lemma 1

Lemma 2. *Each correct process decides a value.*

Proof. As there are at least $n - t$ correct processes that participate in the set agreement problem, no process can block forever at line 04. There two cases.

- If p_i is such that $winners_i \neq \emptyset$ (line 06), it trivially decides at line 10.
- If p_i is such that $winners_i = \emptyset$ (line 06), it executes line 08. Due to Lemma 1, the set set_i is not empty. Consequently, the entry ℓ_i from which p_i decides is well-defined (it does exist).

It follows that each correct process decides. □Lemma 2

Lemma 3. *The number of values that are decided is at most t, and a decided value is a proposed value.*

Proof. If no process ever executes line 05, the agreement and validity property are trivially satisfied. So, let us assume that at least one process executes line 05. Moreover, let RENAMED be the smallest array value obtained by a process when it exits the **repeat** loop at line 04. We consider two cases.

- $\exists x$: $\text{RENAMED}[x] = 1$.
  In that case there is at least one winner, namely, p_x. Due to the containment property, $\text{RENAMED}_i[x] = 1$ for any process p_i that decides. It follows from that observation and the lines 05–06 that any process that decides, does decide the value proposed by a winner process. As at most t processes can obtain a new name comprised between 1 and t (lines 02–03), it follows that there are at most t winners. Consequently, no more than t different values can be decided.
- $\forall x$: $\text{RENAMED}[x] \neq 1$.
  In that case, let $R = \{x : \text{RENAMED}[x] = 0\}$ (hence, all other entries of RENAMED are equal to $\perp$). Due to the exit condition of the **repeat** loop (line 04), we have $|R| \geq n - t$, from which it follows that $|\Pi \setminus R| \leq t$. We claim (claim C) that any process p_i that decides, decides a value proposed by a process p_y such that $y \in \Pi \setminus R$. Combining this claim with $|\Pi \setminus R| \leq t$, we conclude that at most t different values can be decided.
  Proof of the claim C. Let p_i be a process that decides. It decides the value in $PROP[y]$ where y has been determined at line 06 or line 08.
  - p_i selects y at line 06. In that case, p_i decides the value proposed by a process p_y such that $\text{RENAMED}_i[y] = 1$. As $\text{RENAMED} \leq \text{RENAMED}_i$ (snapshot containment property), and RENAMED does not contain the value 1, we conclude that $y \notin R$, and the claim C follows.
  - p_i selects y at line 08. In that case, p_i decides a value proposed by a process p_y such that $\text{RENAMED}_i[y] = \perp$. As $\text{RENAMED}_i[y] = \perp$ and $\text{RENAMED} \leq \text{RENAMED}_i$, we conclude from the definition of R that $y \notin R$, which proves the claim C. □Lemma 3

Theorem 1. *The algorithm described in Fig. 1 is a t-resilient t-set agreement algorithm.*

Proof. The proof follows directly from Lemmas 3 and 2. □Theorem 1

3.3. From k-test&set to k-set agreement

In the k-test&set problem, the processes invoke an operation k_test&set(), and obtain the value 1 (winner) or the value 0 (loser). The values returned to the processes satisfy the following property: there are at least one and at most k winners.

In a very interesting way, the algorithm described in Fig. 1 allows solving the k-set agreement problem from any solution to the k-test&set problem, when $k = t$, $\forall t$. The only modification consists in replacing the lines 02–03 by the following statement: $RENAMED[i] \leftarrow$ k_test&set().

4. An impossibility result

This section proves that, when $k < n/2$, the k-set agreement problem cannot be wait-free solved from atomic registers and a solution to the adaptive $(p + k - 1)$-renaming problem if $k < t$.

Theorem 2. *The k-set agreement problem cannot be wait-free solved (i.e., for $t = n - 1$), in asynchronous systems made up of atomic registers and a solution to the adaptive $(p + k - 1)$-renaming problem, for $n \geq 2k + 1$.*

Notation. The proof uses the following notations:

- f_k: the function $p \to 2p - \lceil \frac{p}{k} \rceil$.
- g_k: the function $p \to \min(2p - 1, p + k - 1)$.
- (n, k)-TS: the k-test&set problem with up to n participating processes. (At least one and most k processes are winners.)
- (n, k)-SA: the k-set agreement problem with up to n participating processes.
- (n, f_k)-AR: the adaptive M-renaming problem with $M = f_k(p)$ (where $p \leq n$ is the number of processes that participate in the renaming).
- (n, g_k)-AR: the adaptive M-renaming problem with $M = g_k(p)$ (where $p \leq n$ is the number of processes that participate in the renaming).
- Any solution to the (n, ℓ)-XX problem (where XX is TS, SA, or AR, and ℓ is k, f_k or g_k) defines a corresponding (n, ℓ)-XX object. So say indifferently "(n, ℓ)-XX problem" or "(n, ℓ)-XX object".

Let us observe that $\forall p, \forall k$, we have $f_1(p) \leq g_k(p)$. This means that any solution to (n, f_1)-AR is a solution to (n, g_k)-AR (Observation $O1$).

Proof. The proof consists in showing the following: $\forall k, \forall n \geq 2k + 1$: there is no wait-free algorithm (i.e. an algorithm working for $t = n - 1$) that solves (n, k)-SA from (n, g_k)-AR. The proof is by contradiction. Let us assume that there is an algorithm $\mathcal{A}$ that solves (n, k)-SA from (n, g_k)-AR with $n \geq 2k+1$. The $(2, 1)$-SA problem is key in proving the contradiction. We have the following.

1. On one hand.
   - The $(2, 1)$-TS problem and the $(2, 1)$-SA problem are equivalent [13].
   - There is a wait-free construction of (n, k)-TS objects from $(2, 1)$-TS objects [13].
   - The (n, f_1)-AR problem can be wait-free solved from $(n, 1)$-TS objects [25].
   - For any $k \geq 1$, the (n, g_k)-AR problem can be wait-free solved from (n, f_1)-AR objects (observation $O1$).
   - Due to the assumption, the algorithm $\mathcal{A}$ solves the (n, k)-SA problem from (n, g_k)-AR objects with $n \geq 2k + 1$, when $t = n - 1$.
   - It follows that, when $t = n - 1$ and $n \geq 2k + 1$, it is possible to solve the (n, k)-SA problem from $(2, 1)$-SA objects.
2. On the other hand.
   - It is shown in [19] that $k \geq j\lfloor \frac{t+1}{m} \rfloor + \min\big(j, (t + 1) \bmod m\big)$ is a necessary requirement for having a t-resilient k-set agreement algorithm for n processes, when these processes share atomic registers and (m, j)-SA objects (objects that allow solving j-set agreement among m processes).
   - Let us consider the case where the (m, j)-SA objects are $(2, 1)$-SA objects. Taking $t = n - 1$, we have $k \geq \lfloor \frac{t+1}{2} \rfloor + \min\big(1, (t + 1) \bmod 2\big)$, from which we obtain the necessary requirement $k \geq \lfloor \frac{n}{2} \rfloor$.
   - It follows that, for $t = n - 1$, $k \geq \lfloor \frac{n}{2} \rfloor$ (i.e., $n \leq 2k$) is a necessary requirement for solving the (n, k)-SA problem from $(2, 1)$-SA objects and atomic registers.
3. The previous items 1 and 2 contradict each other. It follows that the initial assumption (existence of the wait-free algorithm $\mathcal{A}$) cannot hold, which proves the theorem. □Theorem 2

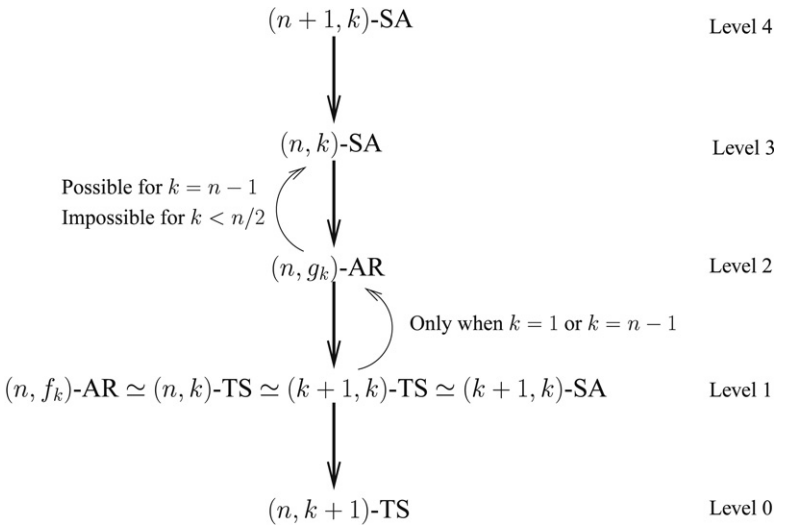

Fig. 3. Towards a wait-free hierarchy for subconsensus problems ($t = n-1$).

5. Conclusion

5.1. The content of the paper

The k-set agreement problem cannot be solved in asynchronous shared memory systems made up of atomic registers only, and where up to t processes may crash, as soon as $k \leq t$. The paper has investigated the additional power provided by an $(p+k-1)$-renaming algorithm when one wants to solve the k-set agreement problem despite $k \leq t$. It has presented two contributions.

The first is a t-resilient algorithm, based on a solution to the adaptive$(p+k-1)$-renaming problem, that solves the k-set agreement problem when $k = t$. The second is an impossibility result showing that, in an asynchronous shared memory system made up of atomic registers and a solution to the $(p+k-1)$-renaming problem, there are values of n for which it is not possible to wait-free solve the k-set agreement problem when $k < t$. Showing this impossibility for any value of n remains an open problem.

For the interested reader, an algorithm building an adaptive $(p+k-1)$-renaming object from a failure detector of the class Ω^k_* is presented in [26]. (This algorithm is a simple extension of an algorithm described in [7] that builds an $(p+k-1)$-renaming object from atomic registers only. The class of failure detectors Ω^k_*, introduced in [28], extends the class of *leader* failure detectors, denoted Ω, that has been introduced in [10].)

5.2. Towards a picture for the wait-free case

The paper has considered the t-resilient case for the design of a k-set agreement algorithm from an adaptive $(p+k-1)$-renaming object (Section 3), and the wait-free case ($t = n-1$) for the lower bound result. This last section presents a global picture for the wait-free case. It depicts a simple hierarchy for the four subconsensus problems that are the adaptive M-renaming problem with $M = f_k(p)$, the adaptive M-renaming problem with $M = g_k(p)$, the k-set agreement problem and the k-test&set problem (more development can be found in [16]).

In addition to the notations introduced in Section 4, we use here the following ones. (x,y)-XX $\succeq$ (x',y')-YY means that there is a wait-free algorithm that solves the (x',y')-YY problem from (x,y)-XX objects and atomic registers. (x,y)-XX $\simeq$ (x',y')-YY means that (x,y)-XX $\succeq$ (x',y')-YY and (x',y')-YY $\succeq$ (x,y)-XX. Let us notice that $f_1 = g_1$, for $p \in [1..n]$: $f_{n-1} = g_{n-1}$ and $g_k < f_k$ when $k \in [2..n-2]$ ([3]) (Observation $O2$).

Global picture. Although the adaptive renaming problem on the one side, and the k-set agreement and k-test&set problems on the other side, seem to be of different nature, they can be ranked in a single hierarchy. More specifically, combining the result of this paper with results of other papers [13,14,19,25] provides us with Fig. 3 that shows that these problems can be ranked in three distinct levels (denoted 1, 2 and 3): (n,k)-SA is stronger than (n,g_k)-AR (this is denoted with a bold arrow), that in turn is stronger than(n,f_k)-AR, (n,k)-TS, $(k+1,k)$-TS, and $(k+1,k)$-SA. Moreover, these four problems are equivalent for any pair (n,k).

Interestingly, it is easy to see that, when $n = k+1$, the levels 3, 2 and 1 of previous hierarchy collapse, and all the problems become equivalent. It is also easy to see that, when $k = 1$ and $n > k+1$, the levels 2 and 1 merge (due to $f_1 = g_1$),

[3] $h_k < \ell_k$ means that $\forall p: \ 1 \leq p \leq n, \ \ h_k(p) \leq \ell_k(p)$ and there is a value of p such that $h_k(p) < \ell_k(p)$.

while the $(n, 1)$-SA problem remains stronger (this follows from the fact that $(n, 1)$-SA is the consensus problem, while the consensus number (definition in Footnote 1) of the $(n, 1)$-TS object is 2 [17]. Except for the slim arrows from level 1 to level 2 and from level 2 to level 3, the hierarchy is strict. More explicitly, we have the following.

- Equivalences. The equivalences stated in level 1 are established in [16,25].
- Bold arrows (going down).
  - From level 4 to level 3: trivial transformation from $(n + 1, k)$-SA to (n, k)-SA.
  - From level 3 to level 2: transformation (n, k)-SA $\succeq (n, g_k)$-AR in [14].
  - From level 2 to level 1: from the fact that $\forall k : \ 1 \leq k \leq n - 1$: $g_k \leq f_k$.
  - From level 1 to level 0: trivial transformation from (n, k)-TS to $(n, k + 1)$-TS.
- Slim arrows (going up).
  - From level 1 to level 2: follows from $f_1 = g_1$ (case $k = 1$) and $f_{n-1} = g_{n-1}$ (case $k = n - 1$).
  - From level 1/2 to level 3: when $n = k + 1$, (n, k)-SA is $(k + 1, k)$-SA (and both are then equivalent to $(k + 1, g_k)$-AR).
- Impossibility.
  - From level 3 (resp., 0) to level 4 (resp., 1): proved in [19].
  - From level 2 to level 3 for $k < n/2$: Theorem 2.
  - From level 1 to level 2 for $k \neq 1, n - 1$: follows from $\forall k : 1 < k < n - 1$: $g_k < f_k$.

Remark. Let us notice that the hierarchy described in Fig. 3 is far from being complete. More research remains to be done in the area of problem equivalence and problem transformation. As an example, the relation linking the (n, k)-SA problem and the $(n - 2, k - 1)$-SA problem is such an intriguing open problem.

Acknowledgments

The authors would like to acknowledge the referees for their constructive comments that helped improve the presentation and the content of the paper.

References

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, N. Shavit, Atomic snapshots of shared memory, Journal of the ACM 40 (4) (1993) 873–890.
[2] Y. Afek, E. Gafni, S. Rajsbaum, M. Raynal, C. Travers, Simultaneous consensus tasks: A tighter characterization of set consensus, in: Proc. 8th Int'l Conference on Distributed Computing and Networking, ICDCN'06, in: LNCS, vol. 4308, Springer Verlag, 2006, pp. 331–341.
[3] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, R. Reischuk, Renaming in an asynchronous environment, Journal of the ACM 37 (3) (1990) 524–548.
[4] H. Attiya, A. Fouren, Polynomial and adaptive long-lived $(2k - 1)$-renaming, in: Proc. 14th Symposium on Distributed Computing, DISC'00, in: LNCS, vol. 1914, 2000, pp. 149–163.
[5] Y. Afek, M. Merritt, Fast, wait-free $(2k - 1)$-renaming, in: Proc. 18th ACM Symposium on Principles of Distributed Computing, PODC'99, ACM Press, 1999, pp. 105–112.
[6] H. Attiya, O. Rachman, Atomic snapshots in $O(n \log n)$ operations, SIAM Journal on Computing 27 (2) (1998) 319–340.
[7] H. Attiya, J. Welch, Distributed Computing: Fundamentals, Simulations and Advanced Topics, 2nd edition, Wiley-Interscience, 2004, 414 pages.
[8] E. Borowsky, E. Gafni, Immediate atomic snapshots and fast renaming, in: Proc. 12th ACM Symposium on Principles of Distributed Computing, PODC'93, 1993, pp. 41–51.
[9] E. Borowsky, E. Gafni, Generalized FLP impossibility results for t-resilient asynchronous computations, in: Proc. 25th ACM Symposium on Theory of Distributed Computing, STOC'93, ACM Press, 1993, pp. 91–100.
[10] T. Chandra, V. Hadzilacos, S. Toueg, The weakest failure detector for solving consensus, Journal of the ACM 43 (4) (1996) 685–722.
[11] S. Chaudhuri, More choices allow more faults: Set consensus problems in totally asynchronous systems, Information and Computation 105 (1993) 132–158.
[12] E. Gafni, Round-by-round failure detectors: Unifying synchrony and asynchrony, in: Proc. 17th ACM Symposium on Principles of Distributed Computing, PODC'99, ACM Press, 1998, pp. 143–152.
[13] E. Gafni, Read–write reductions, in: Proc. 8th Int'l Conference on Distributed Computing and Networking, ICDCN'06, in: LNCS, vol. 4308, Springer Verlag, 2006, pp. 349–354.
[14] E. Gafni, Renaming with k-set consensus: An optimal algorithm in $n + k - 1$ slots, in: Proc. 10th Int'l Conference on Principles of Distributed Systems, OPODIS'06, in: LNCS, vol. 4305, Springer Verlag, 2006, pp. 36–44.
[15] E. Gafni, S. Rajsbaum, M. Herlihy, Subconsensus tasks: Renaming is weaker than set agreement, in: Proc. 20th Int'l Symposium on Distributed Computing, DISC'06, vol. 4167, Springer-Verlag, 2006, pp. 329–338.
[16] E. Gafni, M. Raynal, C. Travers, Test&set, adaptive renaming and set agreement: A guided visit to asynchronous computability, in: 26th IEEE Symposium on Reliable Distributed Systems, SRDS'07, IEEE Computer Society Press, 2007, pp. 93–102.
[17] M.P. Herlihy, Wait-free synchronization, ACM Transactions on Programming Languages and Systems 13 (1) (1991) 124–149.
[18] M.P. Herlihy, L.D. Penso, Tight bounds for k-set agreement with limited scope accuracy failure detectors, Distributed Computing 18 (2) (2005) 157–166.
[19] M.P. Herlihy, S. Rajsbaum, Algebraic spans, Mathematical Structures in Computer Science 10 (4) (2000) 549–573.
[20] M.P. Herlihy, N. Shavit, The topological structure of asynchronous computability, Journal of the ACM 46 (6) (1999) 858–923.
[21] M.P. Herlihy, J.M. Wing, Linearizability: A correctness condition for concurrent objects, ACM Transactions on Programming Languages and Systems 12 (3) (1990) 463–492.
[22] L. Lamport, On interprocess communication, part II: Algorithms, Distributed Computing 1 (2) (1986) 86–101.
[23] A. Mostéfaoui, M. Raynal, k-set agreement with limited accuracy failure detectors, in: Proc. 19th ACM Symposium on Principles of Distributed Computing, PODC'00, ACM Press, 2000, pp. 143–152.
[24] A. Mostéfaoui, M. Raynal, Randomized set agreement, in: Proc. 13th ACM Symposium on Parallel Algorithms and Architectures, SPAA'01, ACM Press, 2001, pp. 291–297.
[25] A. Mostéfaoui, M. Raynal, C. Travers, Exploring Gafni's reduction land: From Ω^k to wait-free adaptive $(2p - \lceil \frac{p}{k} \rceil)$-renaming via k-set agreement, in: Proc. 20th Symposium on Distributed Computing, DISC'06, in: LNCS, vol. 4167, Springer Verlag, 2006, pp. 1–15.
[26] A. Mostéfaoui, M. Raynal, C. Travers, From renaming to k-set agreement, in: 14th Colloquium on Structural Information and Communication Complexity, SIROCCO'07, in: LNCS, vol. 4474, Springer Verlag, 2007, pp. 62–76.
[27] A. Mostéfaoui, M. Raynal, C. Travers, Narrowing power vs efficiency in synchronous set agreement, in: Proc. 9th Int'l Conference on Distributed Computing and Networking, ICDCN'08, in: LNCS, vol. 4904, Springer Verlag, 2008, pp. 99–111.
[28] M. Raynal, C. Travers, In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement, in: Proc. 10th Int'l Conference on Principles of Distributed Systems, OPODIS'06, in: LNCS, vol. 4305, Springer Verlag, 2006, pp. 1–17.
[29] M. Saks, F. Zaharoglou, Wait-free k-set agreement is impossible: The topology of public knowledge, SIAM Journal on Computing 29 (5) (2000) 1449–1483.